US009444752B2

(12) **United States Patent**
**Backholm**

(10) **Patent No.: US 9,444,752 B2**
(45) **Date of Patent: *Sep. 13, 2016**

# (54) DISTRIBUTED CACHING SYSTEMS WITH CONFIGURABLE EXTENDED CACHING OPTIMIZATION

(71) Applicant: **Seven Networks, LLC**, Marshall, TX (US)

(72) Inventor: **Ari Backholm**, Los Altos, CA (US)

(73) Assignee: **Seven Networks, LLC**, Marshall, TX (US)

( * ) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

This patent is subject to a terminal disclaimer.

(21) Appl. No.: **14/502,386**

(22) Filed: **Sep. 30, 2014**

(65) **Prior Publication Data**

US 2015/0016264 A1 Jan. 15, 2015

**Related U.S. Application Data**

(63) Continuation of application No. PCT/US2014/046537, filed on Jul. 14, 2014.

(60) Provisional application No. 61/861,933, filed on Aug. 2, 2013, provisional application No. 61/860,331, filed

(Continued)

(51) **Int. Cl.**
***H04L 12/823*** (2013.01)
***H04L 12/26*** (2006.01)

(Continued)

(52) **U.S. Cl.**
CPC ............... ***H04L 47/32*** (2013.01); ***H04L 43/16*** (2013.01); ***H04L 47/12*** (2013.01);

(Continued)

(58) **Field of Classification Search**
CPC ....... H04L 47/32; H04L 43/16; H04L 47/12; H04L 61/1551; H04L 67/06; H04L 67/2842
USPC .......................................... 370/278, 449, 412
See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

| | | | |
|---|---|---|---|
| 2006/0117249 | A1 | 6/2006 | Hu et al. |
| 2006/0282408 | A1 | 12/2006 | Wisely |

(Continued)

FOREIGN PATENT DOCUMENTS

| | | | |
|---|---|---|---|
| KR | 20130048558 | A | 5/2013 |

OTHER PUBLICATIONS

Non-Final Office Action mailed May 1, 2015 for U.S. Appl. No. 12/080,142.

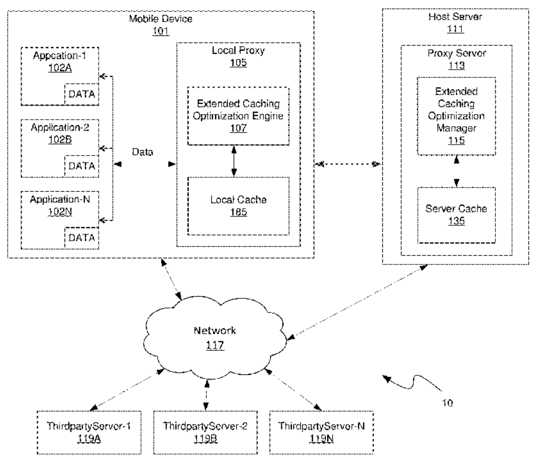
(Continued)

*Primary Examiner* — Ronald B Abelson
(74) *Attorney, Agent, or Firm* — NK Patent Law, PLLC

## (57) ABSTRACT

Methods and systems for distributed caching of information using extended caching optimization are provided. According to one aspect, a method for distributed caching of information using extended caching optimization includes, at a mobile device for operating in a wireless network, monitoring requests issued from an application located within the device to an external entity not located within the device; storing, in a local cache, responses to the monitored requests received from the external entity; and, in response to identifying a request as one that meets a first criterion for optimization, applying an extended caching optimization, including preventing the identified request from being transmitted to the external entity and providing a response to the identified request from the local cache.

**29 Claims, 23 Drawing Sheets**



## Related U.S. Application Data

on Jul. 31, 2013, provisional application No. 61/859,364, filed on Jul. 29, 2013, provisional application No. 61/859,056, filed on Jul. 26, 2013, provisional application No. 61/858,013, filed on Jul. 24, 2013, provisional application No. 61/857,114, filed on Jul. 22, 2013, provisional application No. 61/856,343, filed on Jul. 19, 2013, provisional application No. 61/845,752, filed on Jul. 12, 2013.

(51) **Int. Cl.**

| | |
|---|---|
| ***H04L 12/801*** | (2013.01) |
| ***H04L 29/12*** | (2006.01) |
| ***H04L 29/08*** | (2006.01) |
| ***H04L 29/06*** | (2006.01) |
| ***H04W 28/14*** | (2009.01) |
| ***H04W 4/18*** | (2009.01) |

(52) **U.S. Cl.**
CPC .......... ***H04L 61/1511*** (2013.01); ***H04L 67/06*** (2013.01); ***H04L 67/289*** (2013.01); ***H04L 67/2842*** (2013.01); ***H04L 67/2857*** (2013.01); ***H04L 67/42*** (2013.01); ***H04W 4/18*** (2013.01); ***H04W 28/14*** (2013.01); *H04L 67/1095* (2013.01)

(56) **References Cited**

U.S. PATENT DOCUMENTS

| | | | | |
|---|---|---|---|---|
| 2008/0008095 | A1 | 1/2008 | Gilfix | |
| 2008/0207182 | A1 | 8/2008 | Maharahjh | |
| 2009/0172802 | A1 | 7/2009 | Mosek et al. | |
| 2010/0169407 | A1 | 7/2010 | Hsueh et al. | |
| 2011/0040718 | A1 | 2/2011 | Tendjoukian et al. | |
| 2012/0106415 | A1 | 5/2012 | Li et al. | |
| 2012/0110109 | A1* | 5/2012 | Luna | H04L 67/22 709/213 |
| 2013/0142050 | A1* | 6/2013 | Luna | H04W 4/18 370/241 |
| 2013/0163431 | A1* | 6/2013 | Backholm | H04W 28/0273 370/235 |

OTHER PUBLICATIONS

ISR dated Nov. 24, 2014 for PCT Application PCT/US2014/046537.
Office Action dated Oct. 24, 2013 for U.S. Appl. No. 13/176,537 and associated notice of references cited.
Office Action dated Mar. 8, 2012 for U.S. Appl. No. 13/287,072 and associated notice of references cited.
Office Action dated Feb. 15, 2012 for U.S. Appl. No. 13/287,085 and associated notice of references cited.
Office Action dated Sep. 13, 2012 for U.S. Appl. No. 13/474,561 and associated notice of references cited.
Office Actions dated Mar. 5, 2013, Nov. 8, 2012, and Aug. 15, 2012 for U.S. Appl. No. 13/458,797 and associated notices of references cited.
List of references cited on Nov. 7, 2013 for U.S. Appl. No. 13/592,233.
Office Actions dated May 1, 2014, Oct. 11, 2012, Jul. 2, 2012, and Jan. 3, 2012 for U.S. Appl. No. 13/274,265 and associated notices of references cited.
Office Actions dated Jun. 11, 2014, Jun. 28, 2012, and Jan. 9, 2012 for U.S. Appl. No. 13/274,501 and associated notices of references cited.
Office Action dated Nov. 29, 2013 for U.S. Appl. No. 13/735,868 and associated notice of references cited.
Office Action dated Oct. 8, 2014 for U.S. Appl. No. 13/912,067 and associated notice of references cited.
Office Action dated May 3, 2012 for U.S. Appl. No. 13/274,250 and associated notice of references cited.
Office Actions dated Sep. 11, 2014, Apr. 29, 2014, Oct. 8, 2013, and Nov. 14, 2012 for U.S. Appl. No. 13/572,598 and associated notices of references cited.
Office Actions dated Dec. 13, 2013 and U.S. Appl. No. 13/860,332 for U.S. Appl. No. 13/860,332 and associated notices of references cited.
Non-final rejection mailed Jul. 31, 2015 for U.S. Appl. No. 13/915,538.
WIPO, International Preliminary Report on Patentability for PCT patent application PCT/US2014/046537, Jan. 1, 2016.

* cited by examiner

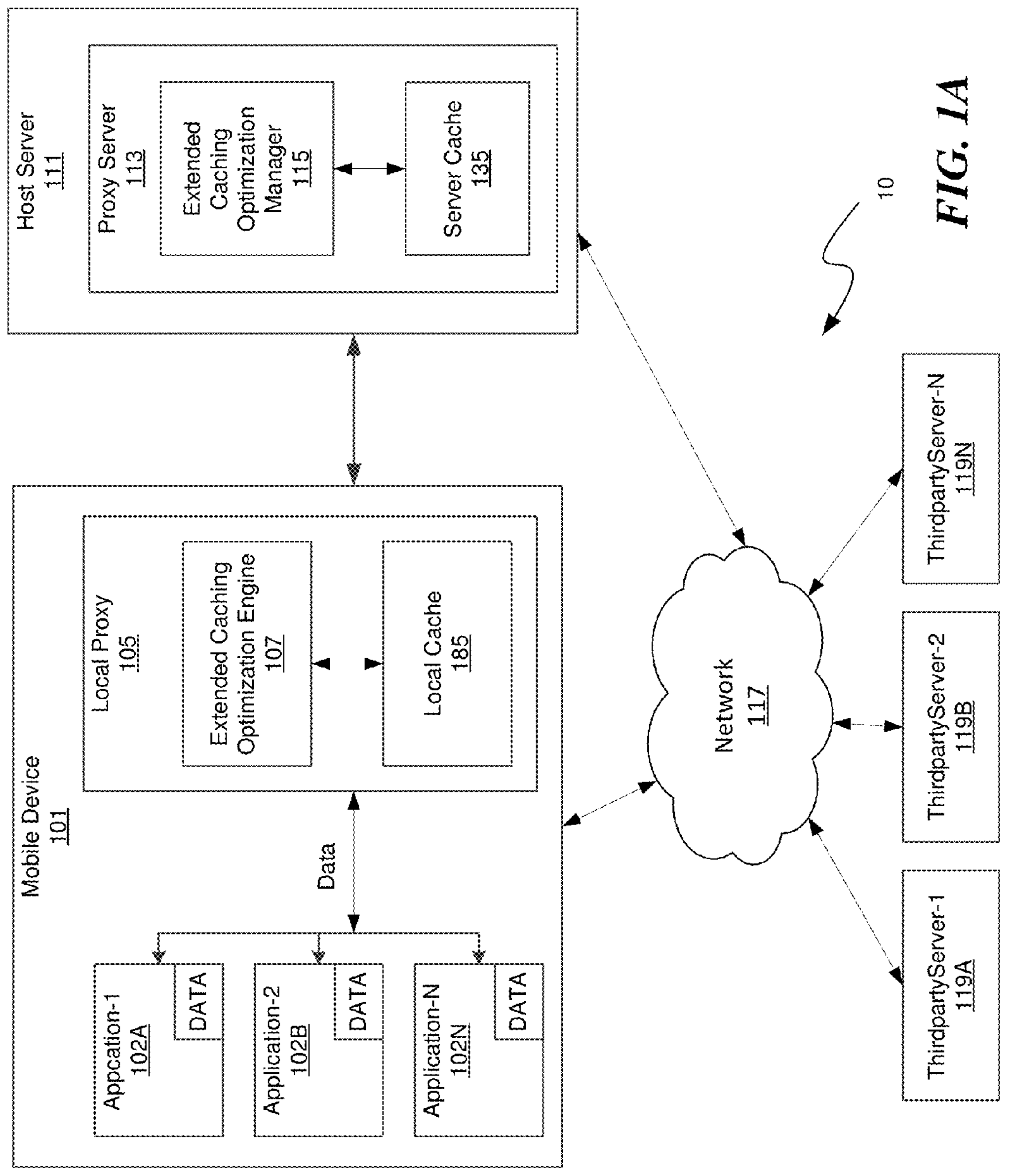

Host Server 111
Proxy Server 113
Extended Caching Optimization Manager 115
Server Cache 135
Mobile Device 101
Local Proxy 105
Extended Caching Optimization Engine 107
Local Cache 185
Data
Appcation-1 102A
DATA
Application-2 102B
DATA
Application-N 102N
DATA
Network 117
ThirdpartyServer-1 119A
ThirdpartyServer-2 119B
ThirdpartyServer-N 119N
10

FIG. 1A

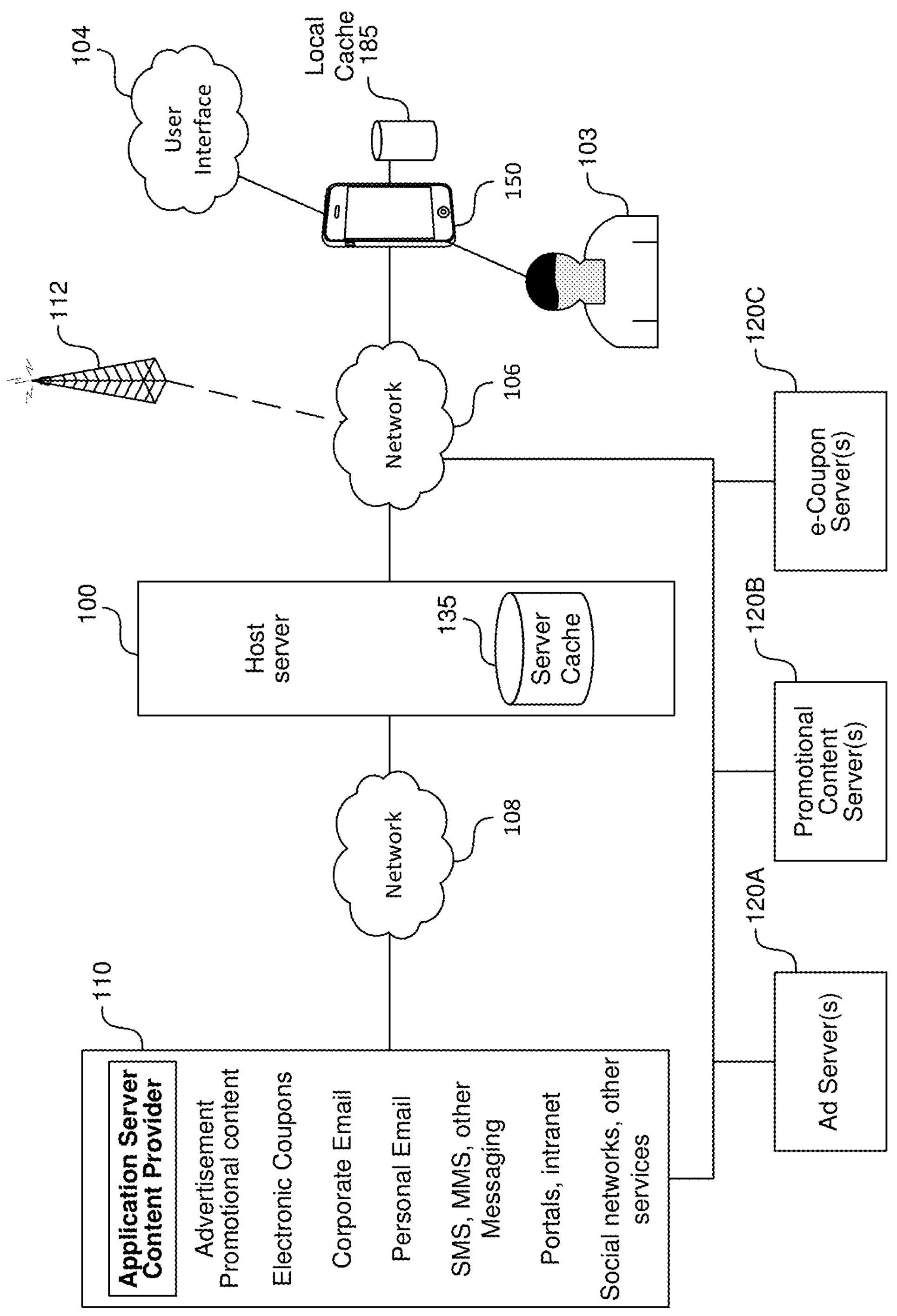

104
User Interface
Local Cache 185
150
103
112
Network
106
120C
e-Coupon Server(s)
100
Host server
135
Server Cache
120B
Promotional Content Server(s)
Network
108
120A
Ad Server(s)
110
Application Server Content Provider
Advertisement Promotional content
Electronic Coupons
Corporate Email
Personal Email
SMS, MMS, other Messaging
Portals, intranet
Social networks, other services

*FIG. 1B*

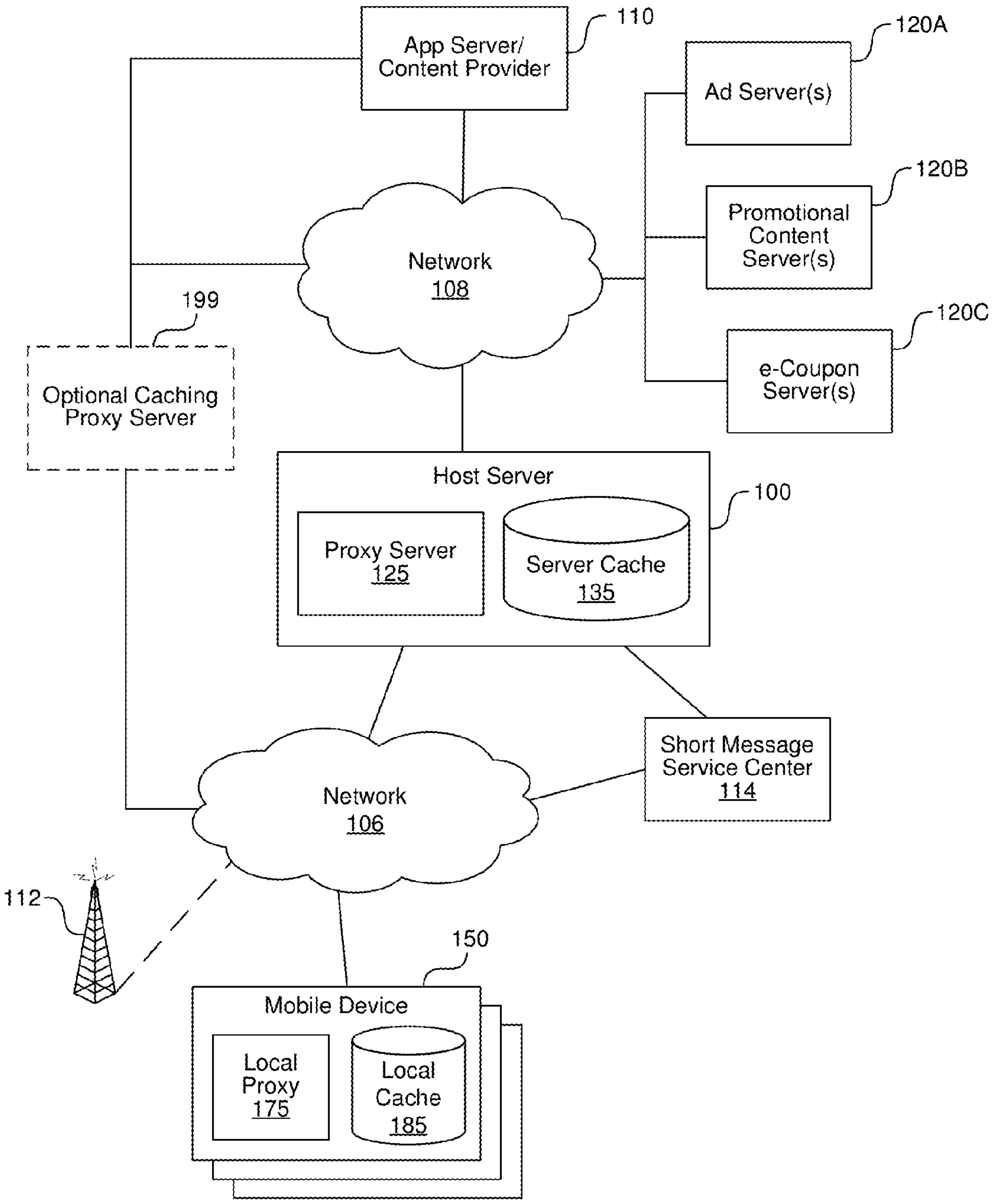

App Server/ Content Provider
110
Ad Server(s)
120A
Promotional Content Server(s)
120B
e-Coupon Server(s)
120C
Network 108
199
Optional Caching Proxy Server
Host Server
100
Proxy Server 125
Server Cache 135
Network 106
Short Message Service Center 114
112
150
Mobile Device
Local Proxy 175
Local Cache 185

FIG. 1C

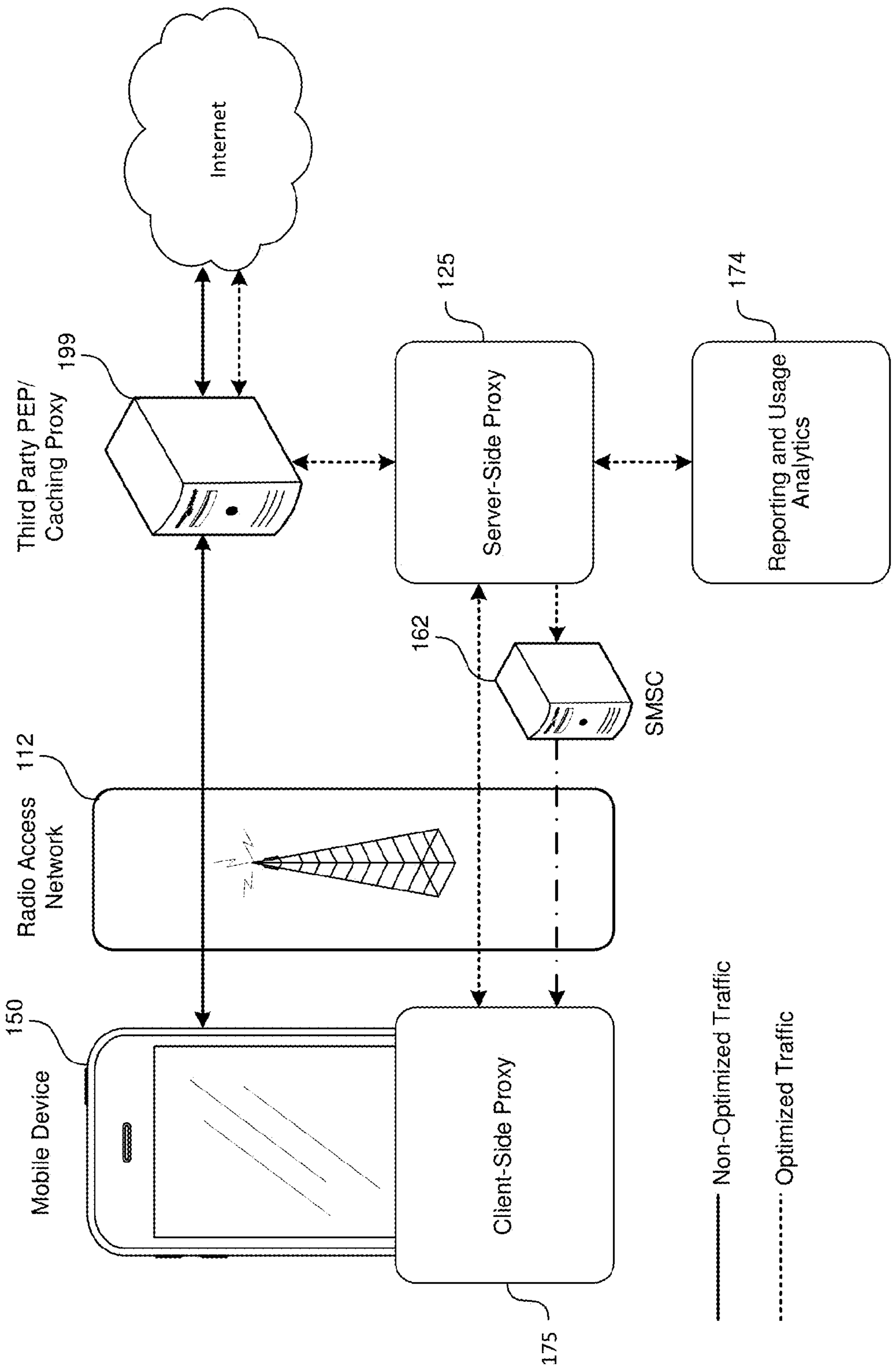

Internet
Third Party PEP/ Caching Proxy
199
125
Server-Side Proxy
174
Reporting and Usage Analytics
162
SMSC
112
Radio Access Network
150
Mobile Device
Client-Side Proxy
175
Non-Optimized Traffic
Optimized Traffic

*FIG. 1D*

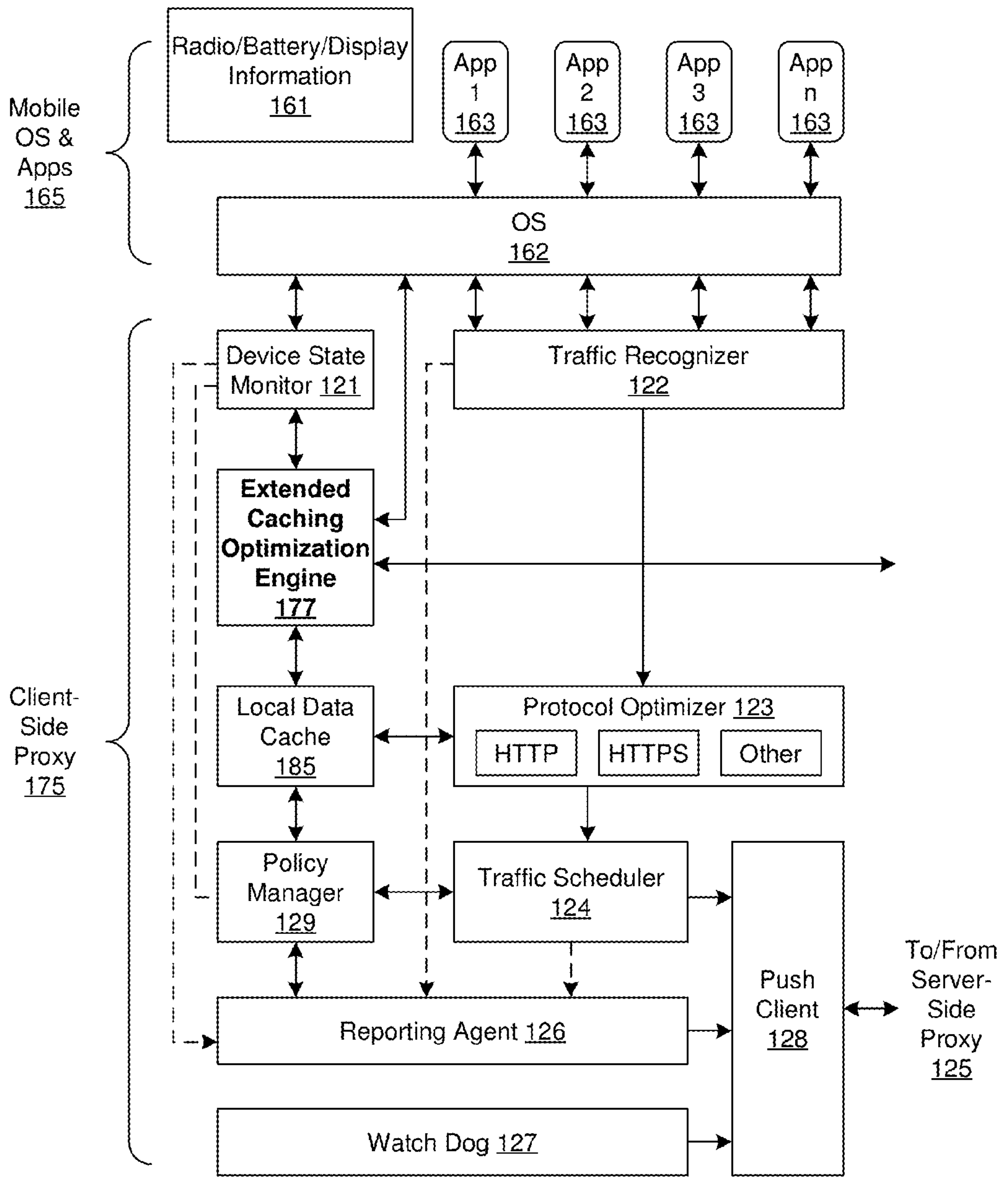

Mobile OS & Apps 165
Radio/Battery/Display Information 161
App 1 163
App 2 163
App 3 163
App n 163
OS 162
Client-Side Proxy 175
Device State Monitor 121
Traffic Recognizer 122
Extended Caching Optimization Engine 177
Local Data Cache 185
Protocol Optimizer 123
HTTP
HTTPS
Other
Policy Manager 129
Traffic Scheduler 124
Push Client 128
To/From Server-Side Proxy 125
Reporting Agent 126
Watch Dog 127

FIG. 1E

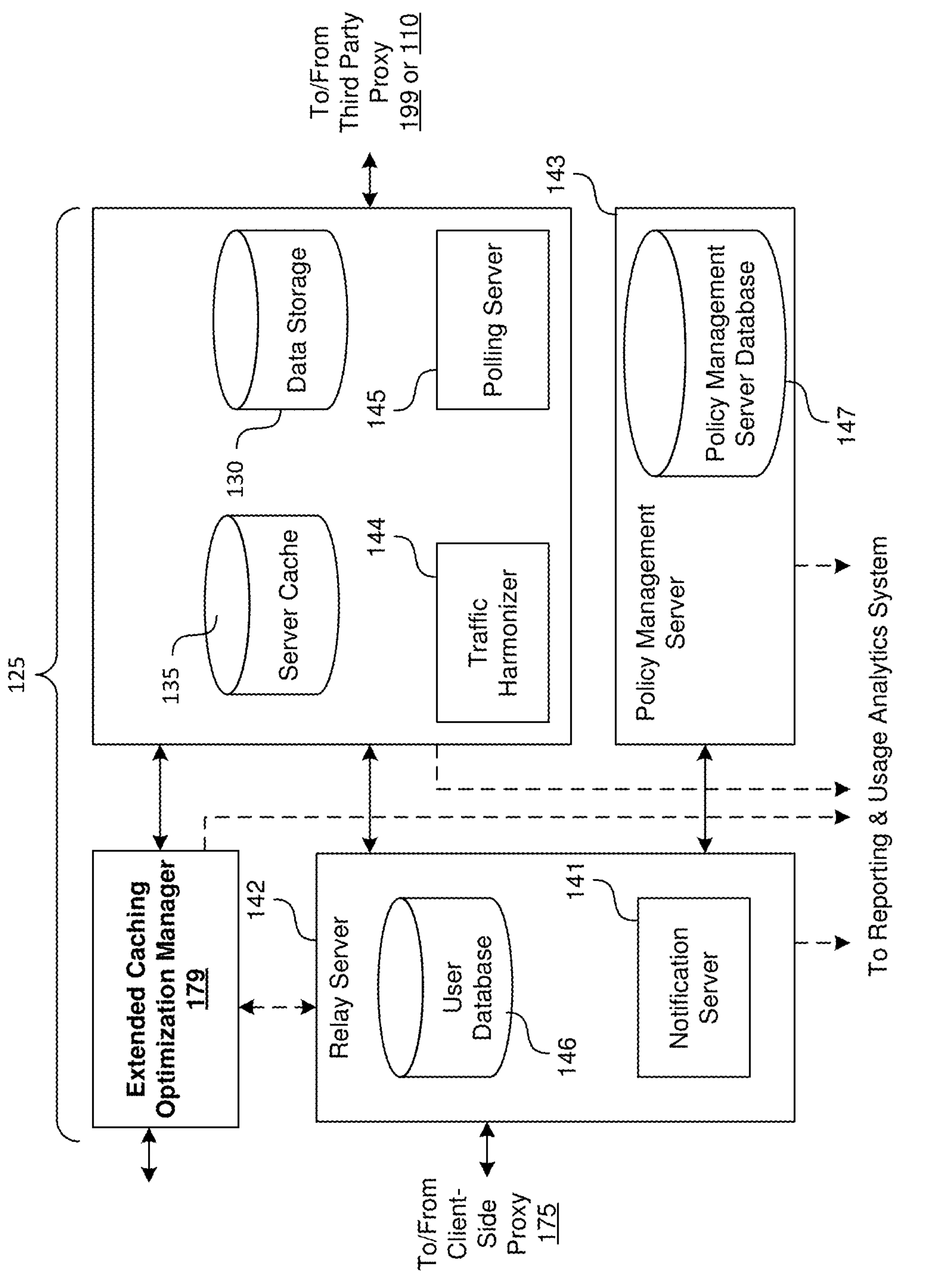

To/From Third Party Proxy 199 or 110
125
135
Server Cache
130
Data Storage
144
Traffic Harmonizer
145
Polling Server
143
Policy Management Server
Policy Management Server Database
147
Extended Caching Optimization Manager 179
142
Relay Server
User Database
146
141
Notification Server
To Reporting & Usage Analytics System
To/From Client-Side Proxy 175

FIG. 1F

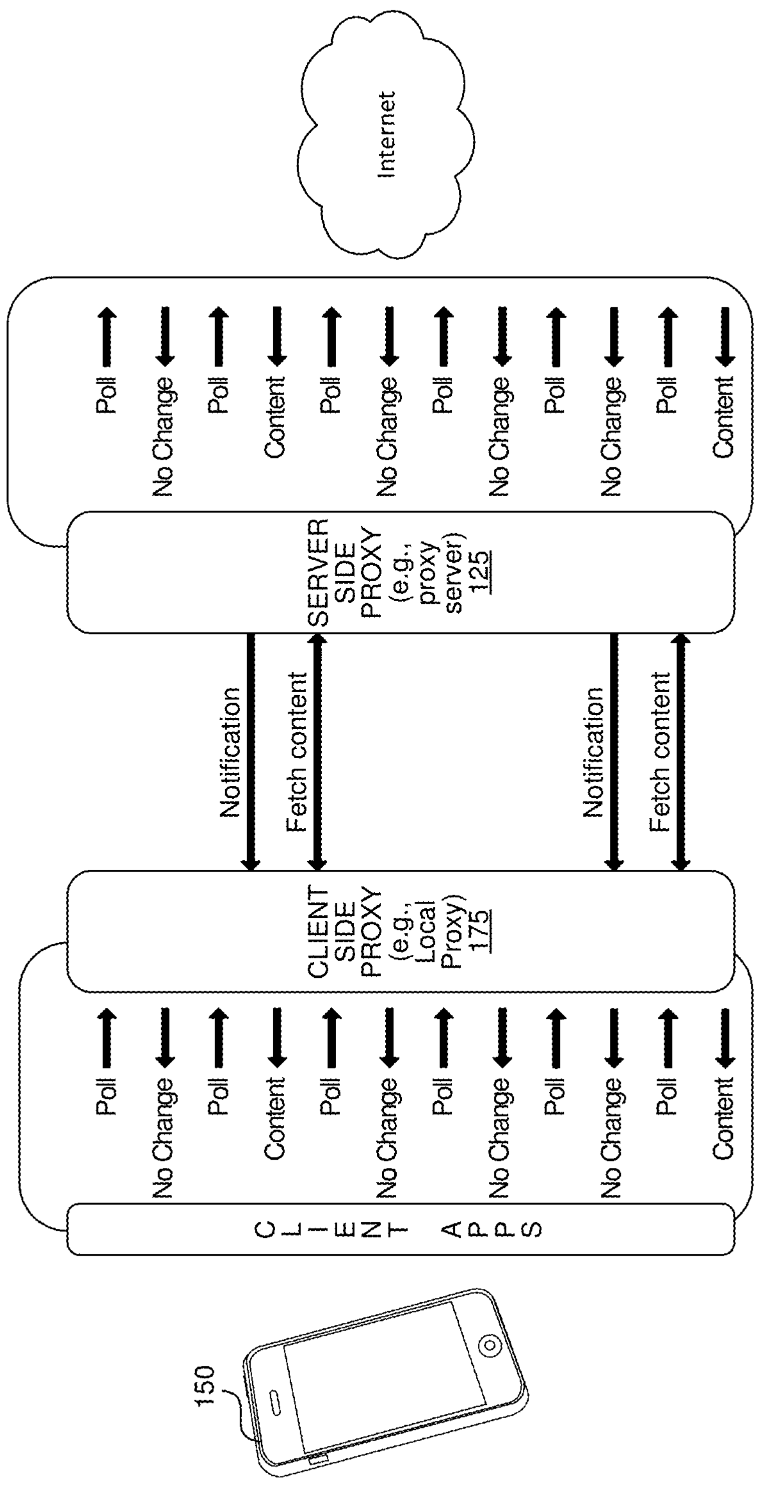

Internet
Poll
No Change
Poll
Content
Poll
No Change
Poll
No Change
Poll
No Change
Poll
Content
SERVER SIDE PROXY (e.g., proxy server) 125
Notification
Fetch content
Notification
Fetch content
CLIENT SIDE PROXY (e.g., Local Proxy) 175
Poll
No Change
Poll
Content
Poll
No Change
Poll
No Change
Poll
No Change
Poll
Content
CLIENT APPS
150

*FIG. 1G*

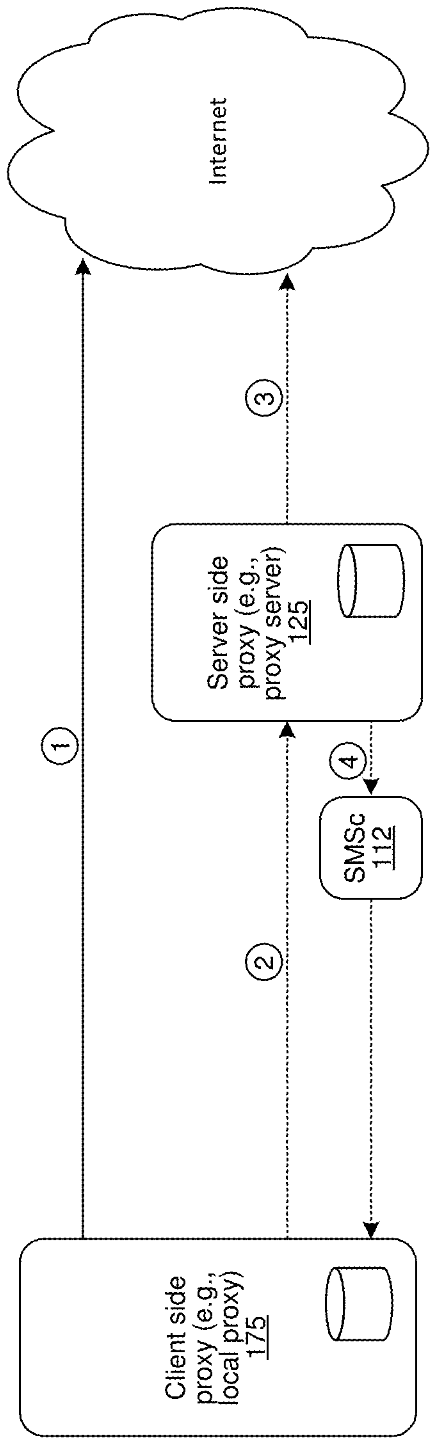

Internet
1
2
3
4
Server side proxy (e.g., proxy server) 125
SMSc 112
Client side proxy (e.g., local proxy) 175

FIG. 1H

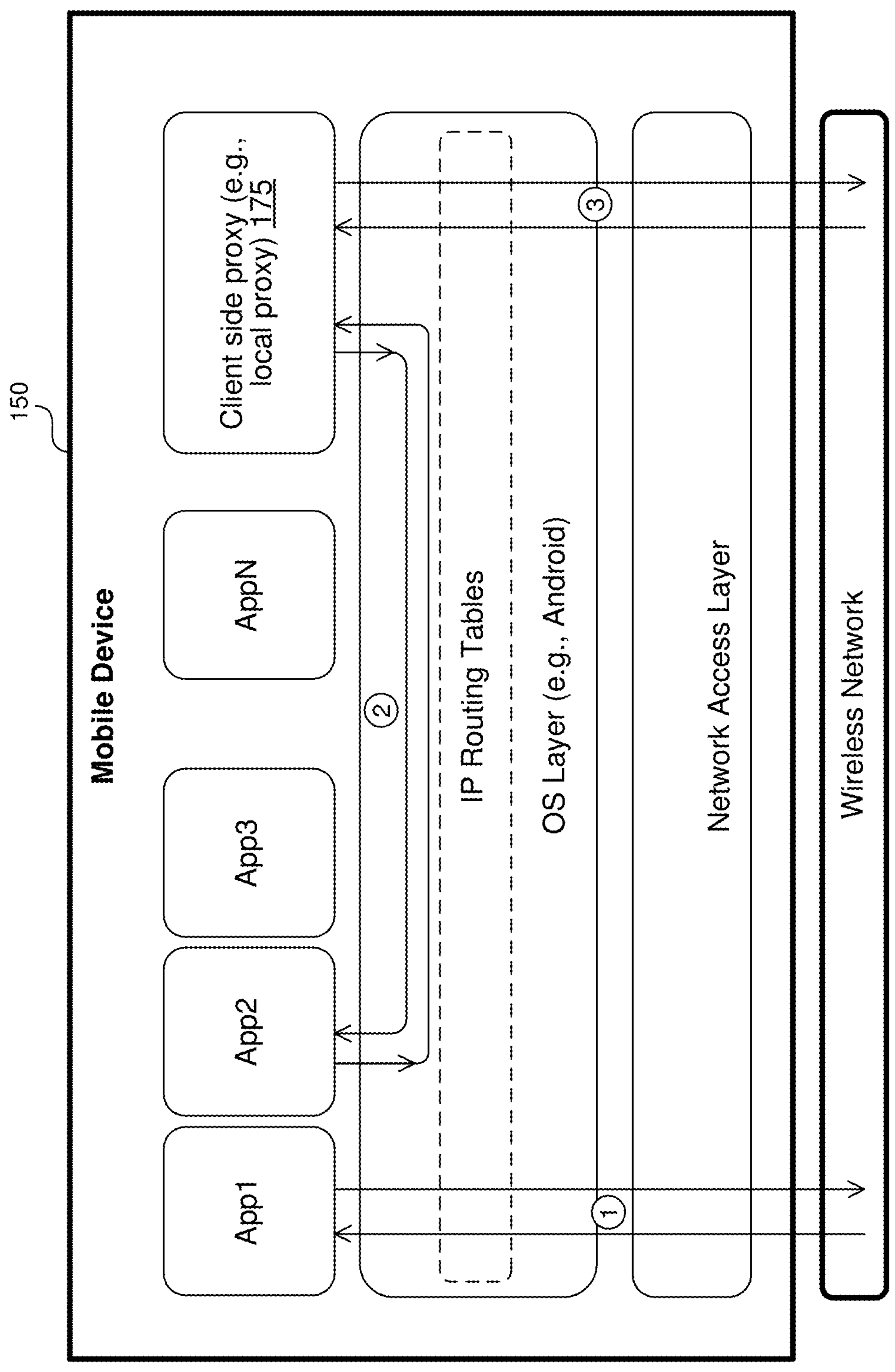

150
Mobile Device
App1
App2
App3
AppN
Client side proxy (e.g., local proxy) 175
IP Routing Tables
OS Layer (e.g., Android)
Network Access Layer
Wireless Network
1
2
3

FIG. 11

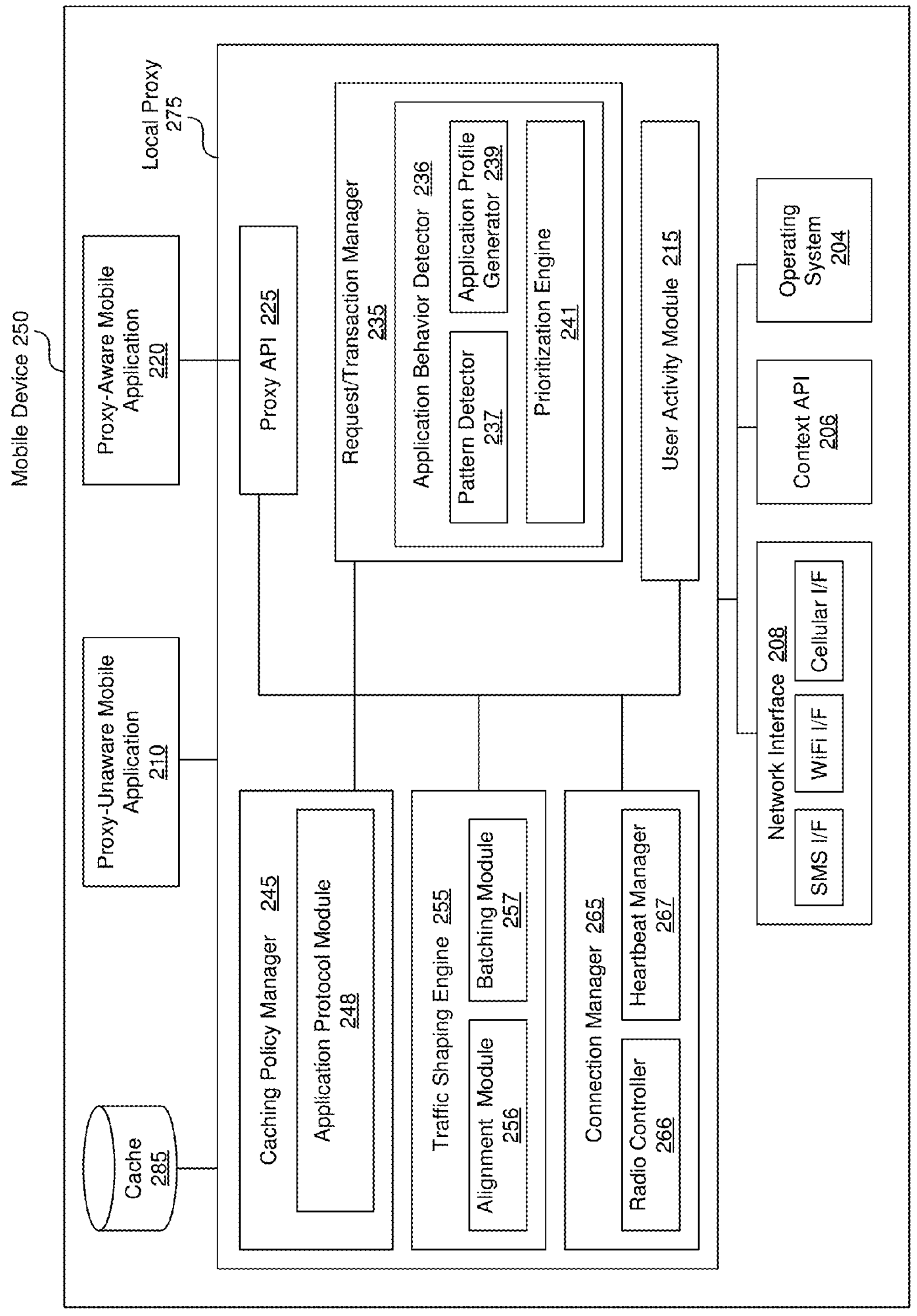

Mobile Device 250
Proxy-Aware Mobile Application 220
Proxy-Unaware Mobile Application 210
Cache 285
Local Proxy 275
Proxy API 225
Request/Transaction Manager 235
Application Behavior Detector 236
Pattern Detector 237
Application Profile Generator 239
Prioritization Engine 241
User Activity Module 215
Caching Policy Manager 245
Application Protocol Module 248
Traffic Shaping Engine 255
Alignment Module 256
Batching Module 257
Connection Manager 265
Radio Controller 266
Heartbeat Manager 267
Operating System 204
Context API 206
Network Interface 208
SMS I/F
WiFi I/F
Cellular I/F

*FIG. 2A*

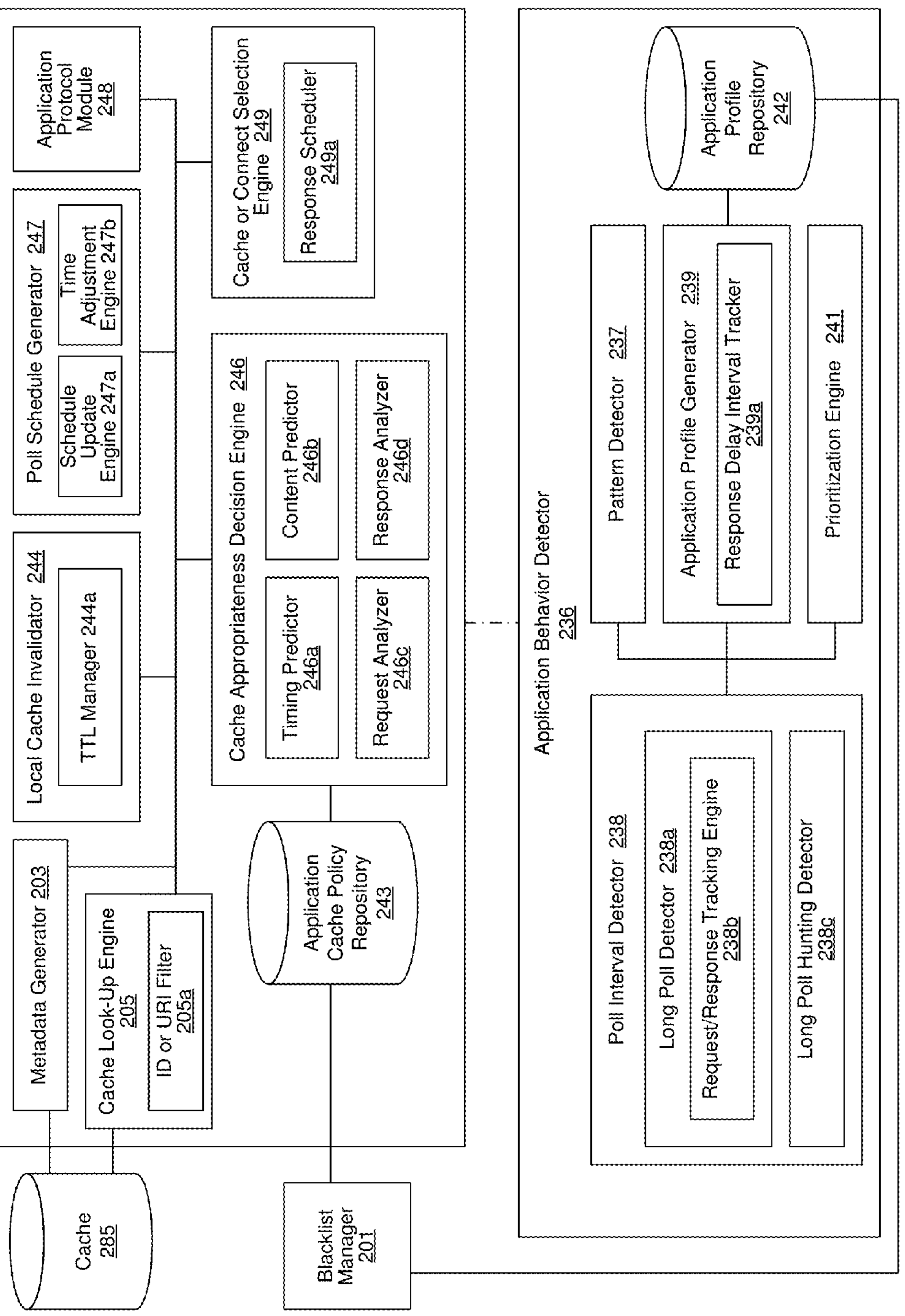

Caching Policy Manager 245
Metadata Generator 203
Cache Look-Up Engine 205
ID or URI Filter 205a
Local Cache Invalidator 244
TTL Manager 244a
Poll Schedule Generator 247
Schedule Update Engine 247a
Time Adjustment Engine 247b
Application Protocol Module 248
Cache Appropriateness Decision Engine 246
Timing Predictor 246a
Content Predictor 246b
Request Analyzer 246c
Response Analyzer 246d
Cache or Connect Selection Engine 249
Response Scheduler 249a
Application Cache Policy Repository 243
Cache 285
Blacklist Manager 201
Application Behavior Detector 236
Poll Interval Detector 238
Long Poll Detector 238a
Request/Response Tracking Engine 238b
Long Poll Hunting Detector 238c
Pattern Detector 237
Application Profile Generator 239
Response Delay Interval Tracker 239a
Prioritization Engine 241
Application Profile Repository 242

FIG. 2B

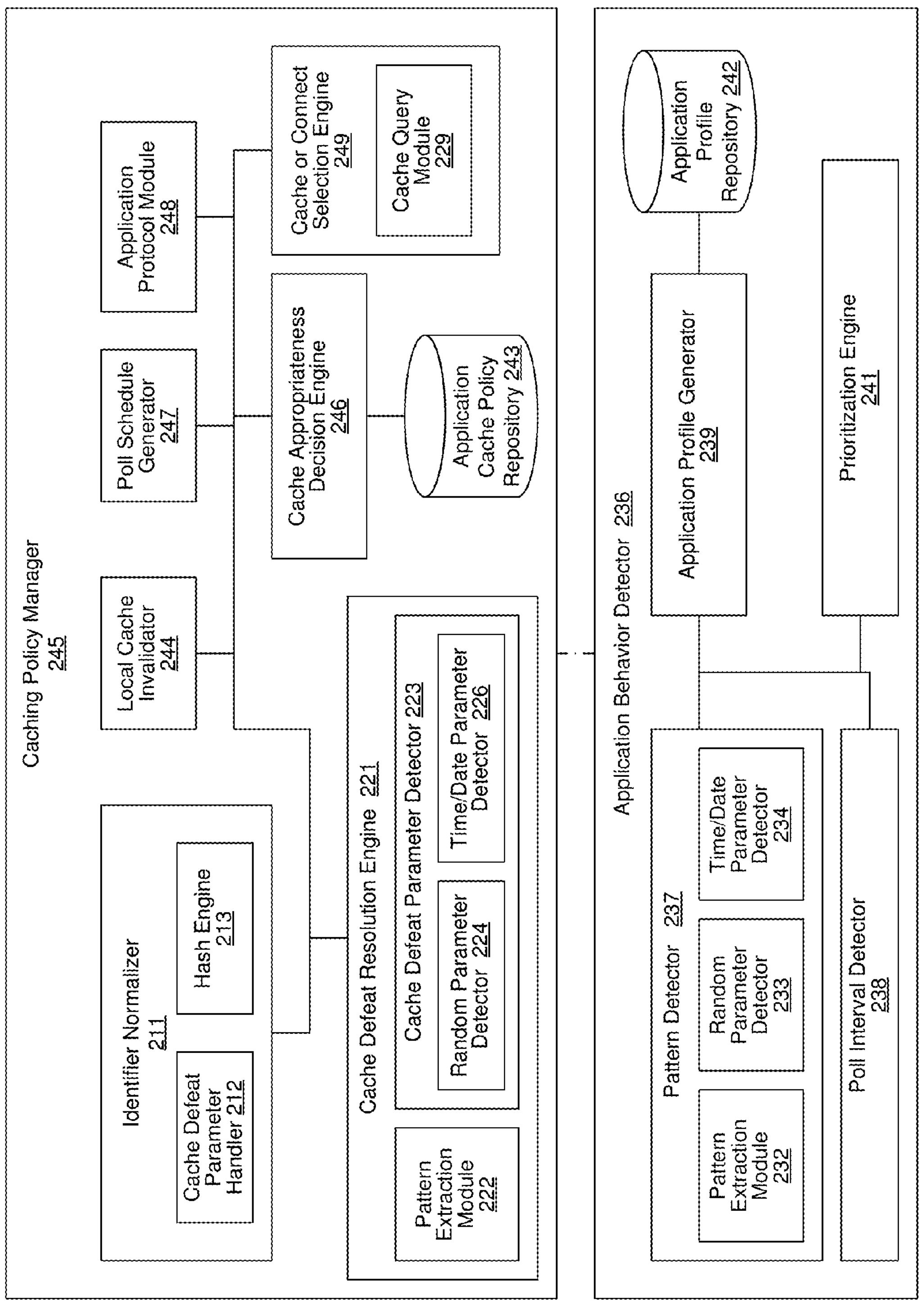

Caching Policy Manager 245
Identifier Normalizer 211
Cache Defeat Parameter Handler 212
Hash Engine 213
Local Cache Invalidator 244
Poll Schedule Generator 247
Application Protocol Module 248
Cache Defeat Resolution Engine 221
Pattern Extraction Module 222
Cache Defeat Parameter Detector 223
Random Parameter Detector 224
Time/Date Parameter Detector 226
Cache Appropriateness Decision Engine 246
Application Cache Policy Repository 243
Cache or Connect Selection Engine 249
Cache Query Module 229
Application Behavior Detector 236
Pattern Detector 237
Pattern Extraction Module 232
Random Parameter Detector 233
Time/Date Parameter Detector 234
Poll Interval Detector 238
Application Profile Generator 239
Prioritization Engine 241
Application Profile Repository 242

FIG. 2C

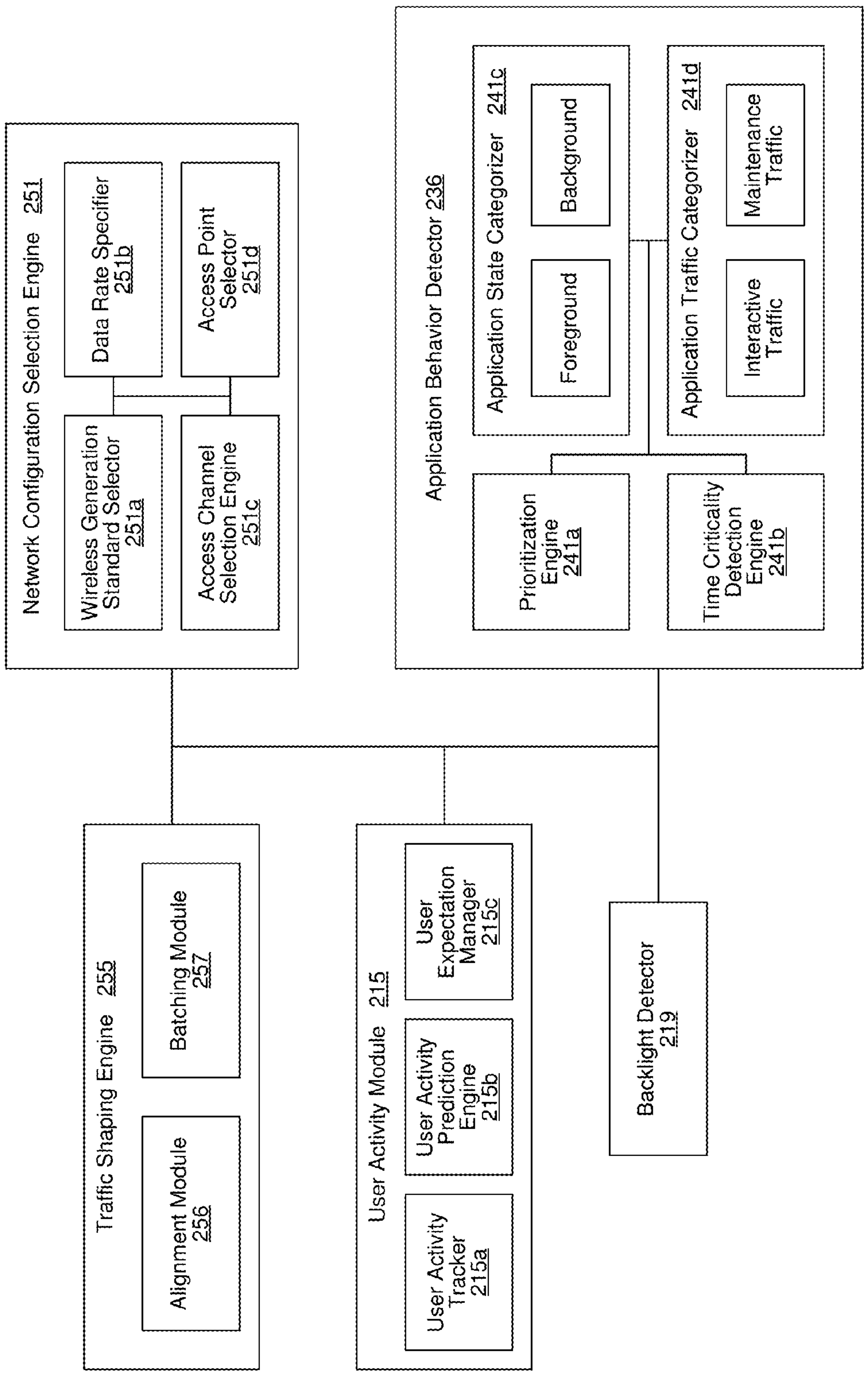

Network Configuration Selection Engine 251
Wireless Generation Standard Selector 251a
Data Rate Specifier 251b
Access Channel Selection Engine 251c
Access Point Selector 251d
Application Behavior Detector 236
Prioritization Engine 241a
Time Criticality Detection Engine 241b
Application State Categorizer 241c
Foreground
Background
Application Traffic Categorizer 241d
Interactive Traffic
Maintenance Traffic
Traffic Shaping Engine 255
Alignment Module 256
Batching Module 257
User Activity Module 215
User Activity Tracker 215a
User Activity Prediction Engine 215b
User Expectation Manager 215c
Backlight Detector 219

FIG. 2D

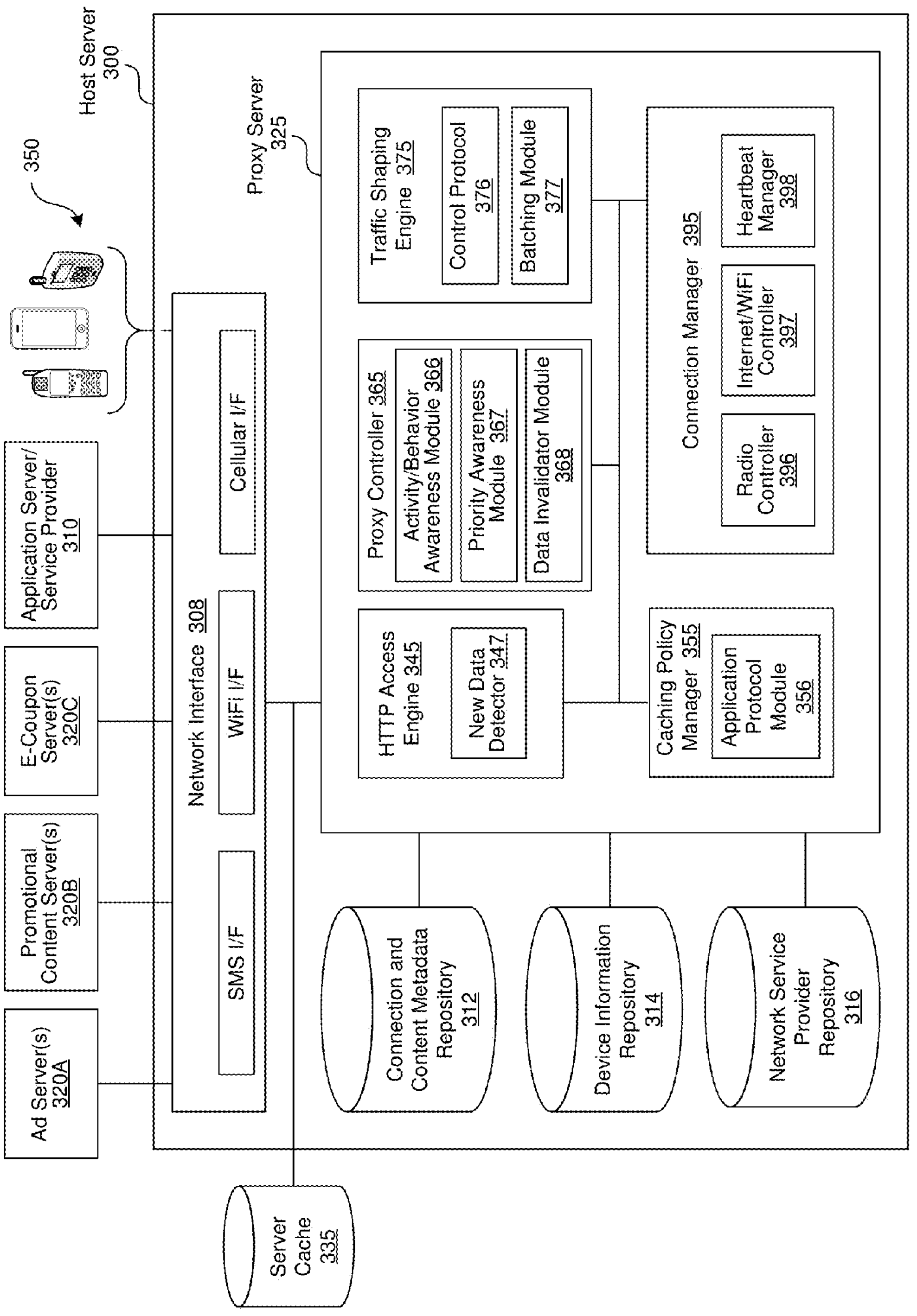

Host Server 300
350
Proxy Server 325
Ad Server(s) 320A
Promotional Content Server(s) 320B
E-Coupon Server(s) 320C
Application Server/ Service Provider 310
Network Interface 308
SMS I/F
WiFi I/F
Cellular I/F
HTTP Access Engine 345
New Data Detector 347
Caching Policy Manager 355
Application Protocol Module 356
Proxy Controller 365
Activity/Behavior Awareness Module 366
Priority Awareness Module 367
Data Invalidator Module 368
Connection Manager 395
Radio Controller 396
Internet/WiFi Controller 397
Heartbeat Manager 398
Traffic Shaping Engine 375
Control Protocol 376
Batching Module 377
Connection and Content Metadata Repository 312
Device Information Repository 314
Network Service Provider Repository 316
Server Cache 335

FIG. 3A

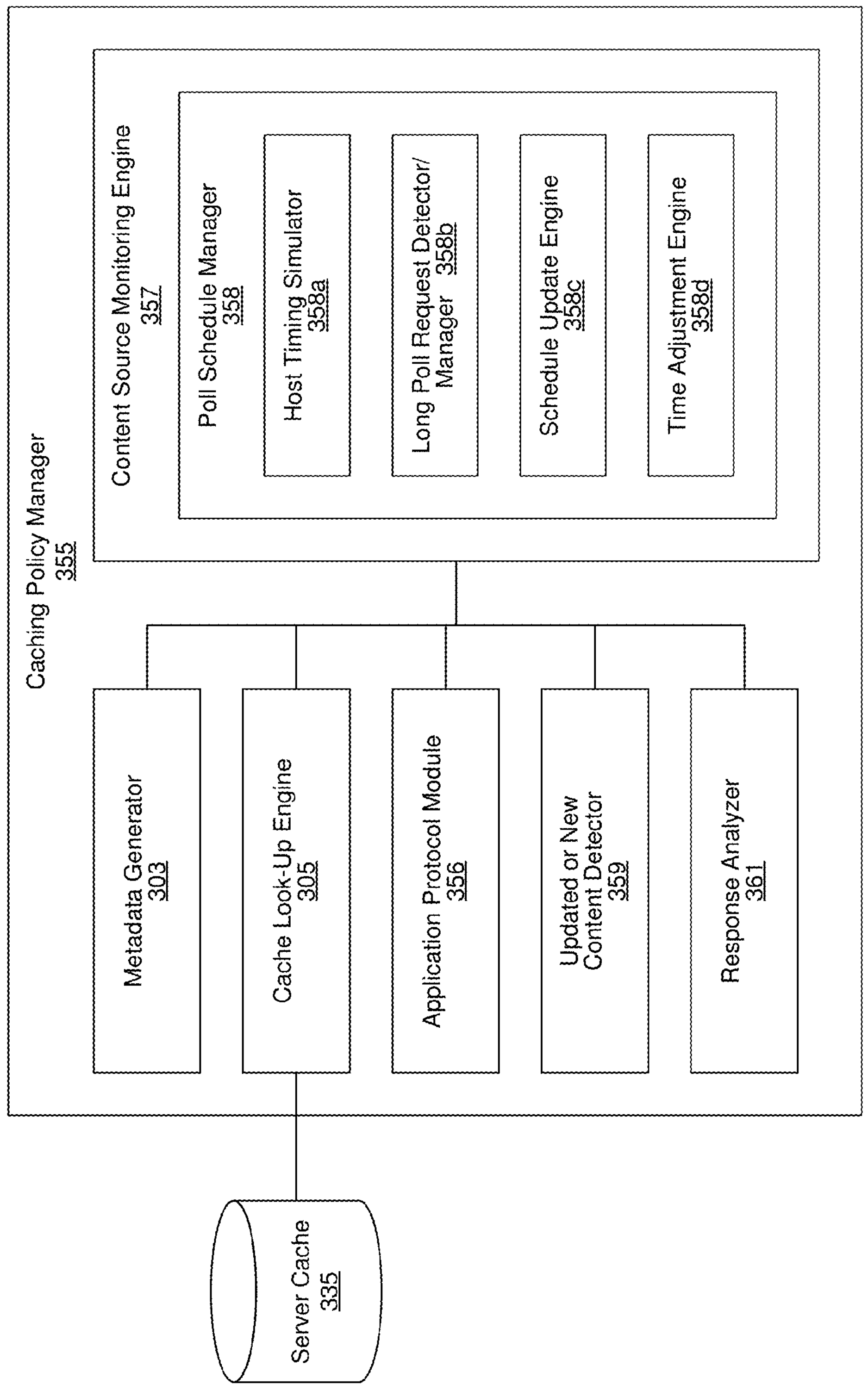

Caching Policy Manager 355
Content Source Monitoring Engine 357
Poll Schedule Manager 358
Host Timing Simulator 358a
Long Poll Request Detector/ Manager 358b
Schedule Update Engine 358c
Time Adjustment Engine 358d
Metadata Generator 303
Cache Look-Up Engine 305
Application Protocol Module 356
Updated or New Content Detector 359
Response Analyzer 361
Server Cache 335

*FIG. 3B*

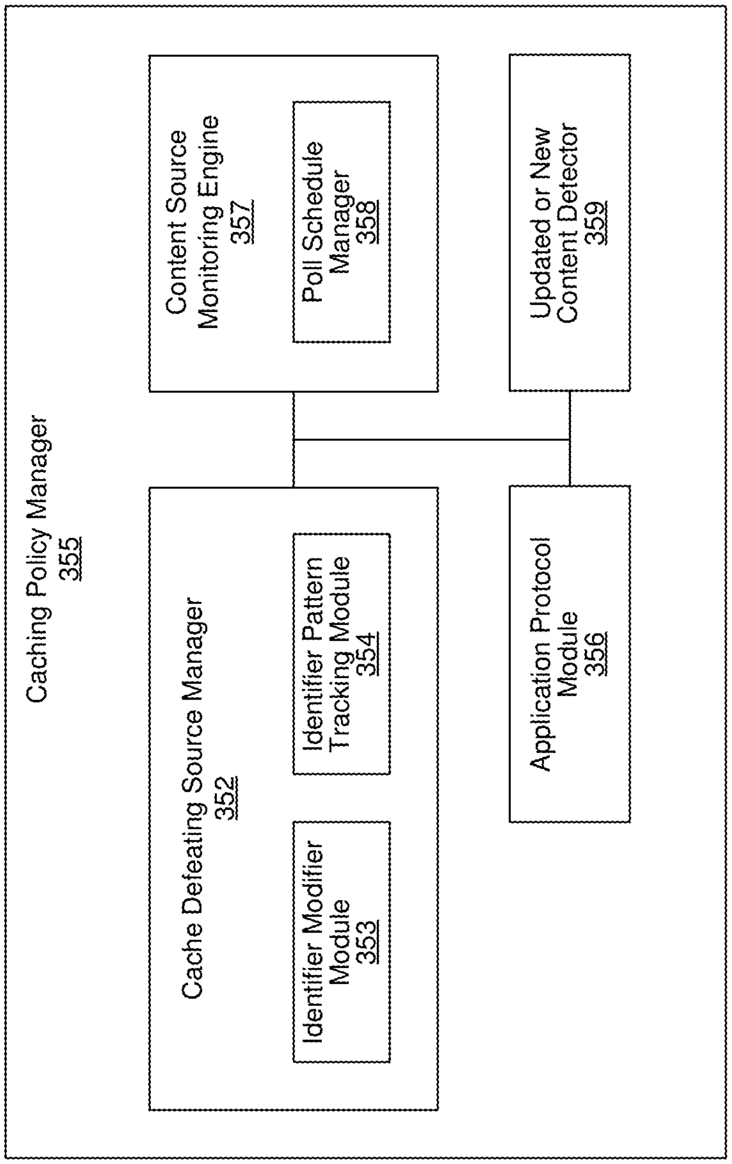

Caching Policy Manager 355
Cache Defeating Source Manager 352
Identifier Modifier Module 353
Identifier Pattern Tracking Module 354
Content Source Monitoring Engine 357
Poll Schedule Manager 358
Application Protocol Module 356
Updated or New Content Detector 359

FIG. 3C

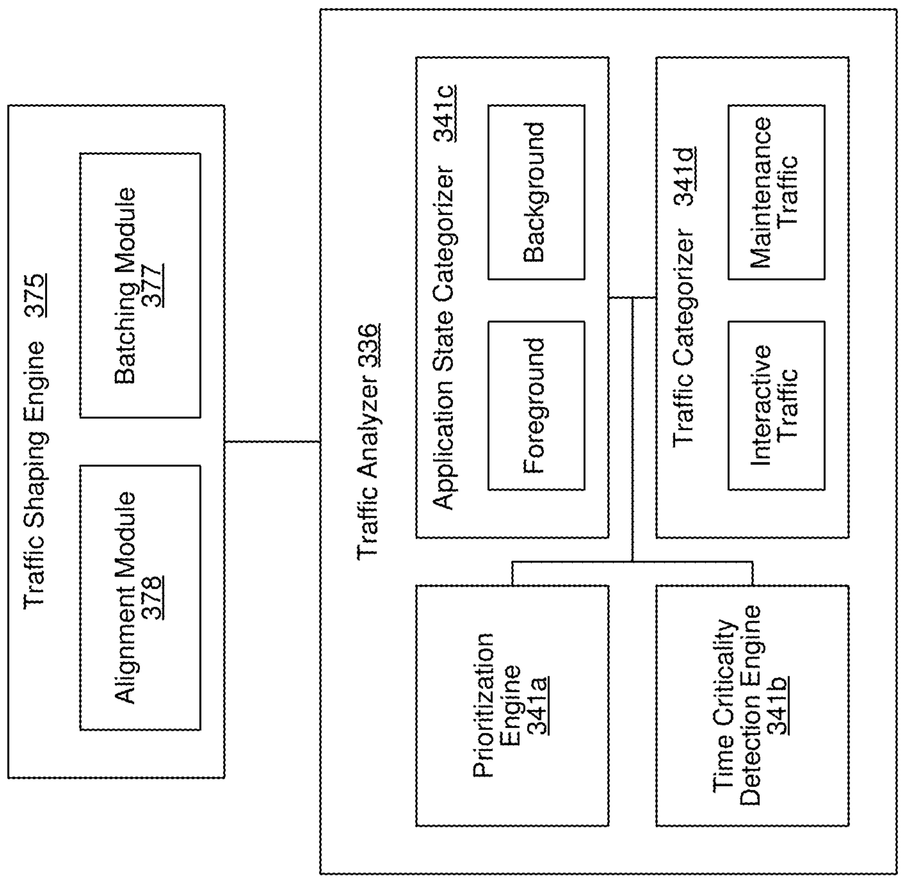

Traffic Shaping Engine 375
Alignment Module 378
Batching Module 377
Traffic Analyzer 336
Prioritization Engine 341a
Time Criticality Detection Engine 341b
Application State Categorizer 341c
Foreground
Background
Traffic Categorizer 341d
Interactive Traffic
Maintenance Traffic

*FIG. 3D*

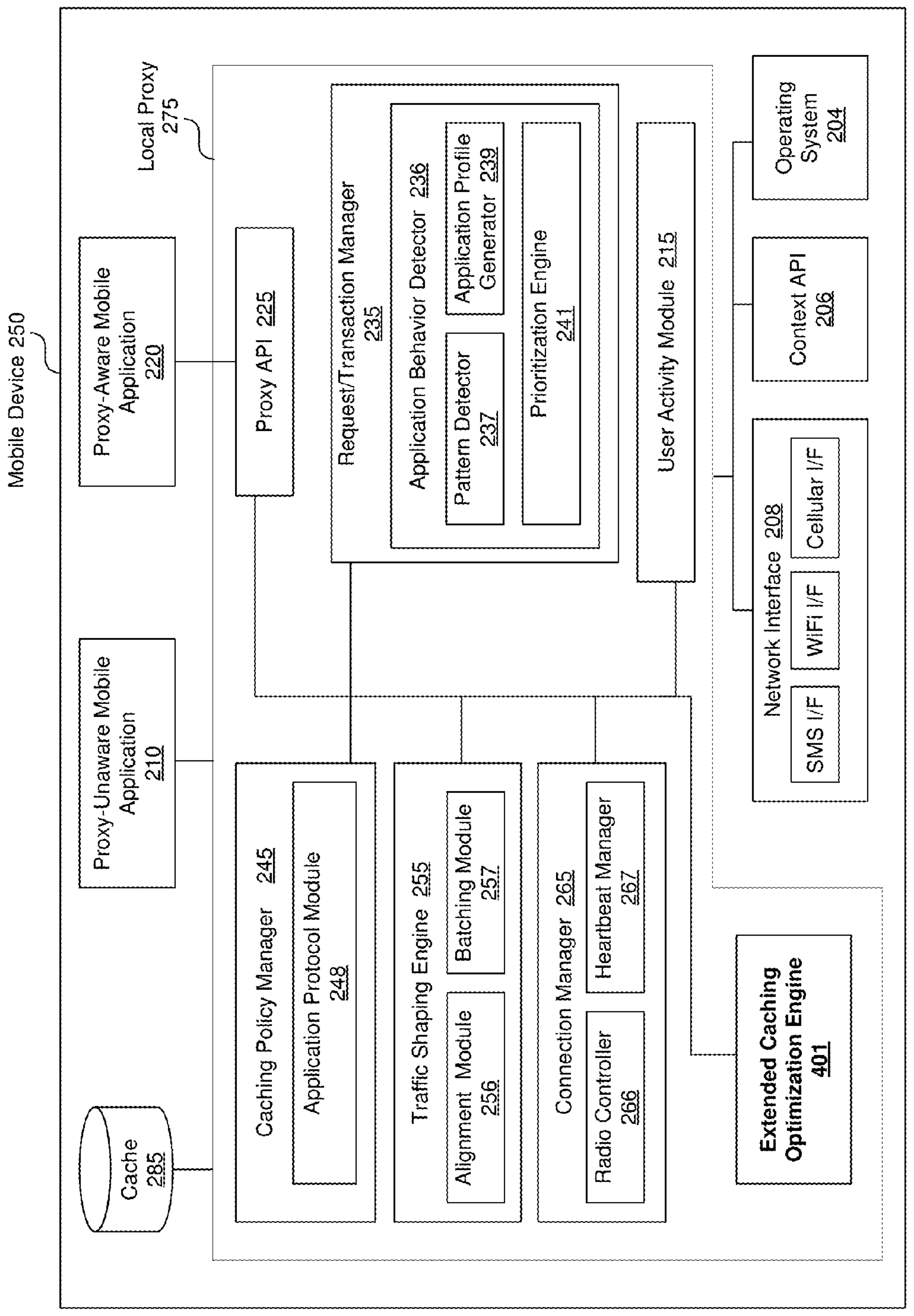

Mobile Device 250
Local Proxy 275
Proxy-Aware Mobile Application 220
Proxy-Unaware Mobile Application 210
Cache 285
Proxy API 225
Request/Transaction Manager 235
Application Behavior Detector 236
Pattern Detector 237
Application Profile Generator 239
Prioritization Engine 241
User Activity Module 215
Caching Policy Manager 245
Application Protocol Module 248
Traffic Shaping Engine 255
Alignment Module 256
Batching Module 257
Connection Manager 265
Radio Controller 266
Heartbeat Manager 267
Extended Caching Optimization Engine 401
Operating System 204
Context API 206
Network Interface 208
SMS I/F
WiFI I/F
Cellular I/F

FIG. 4A

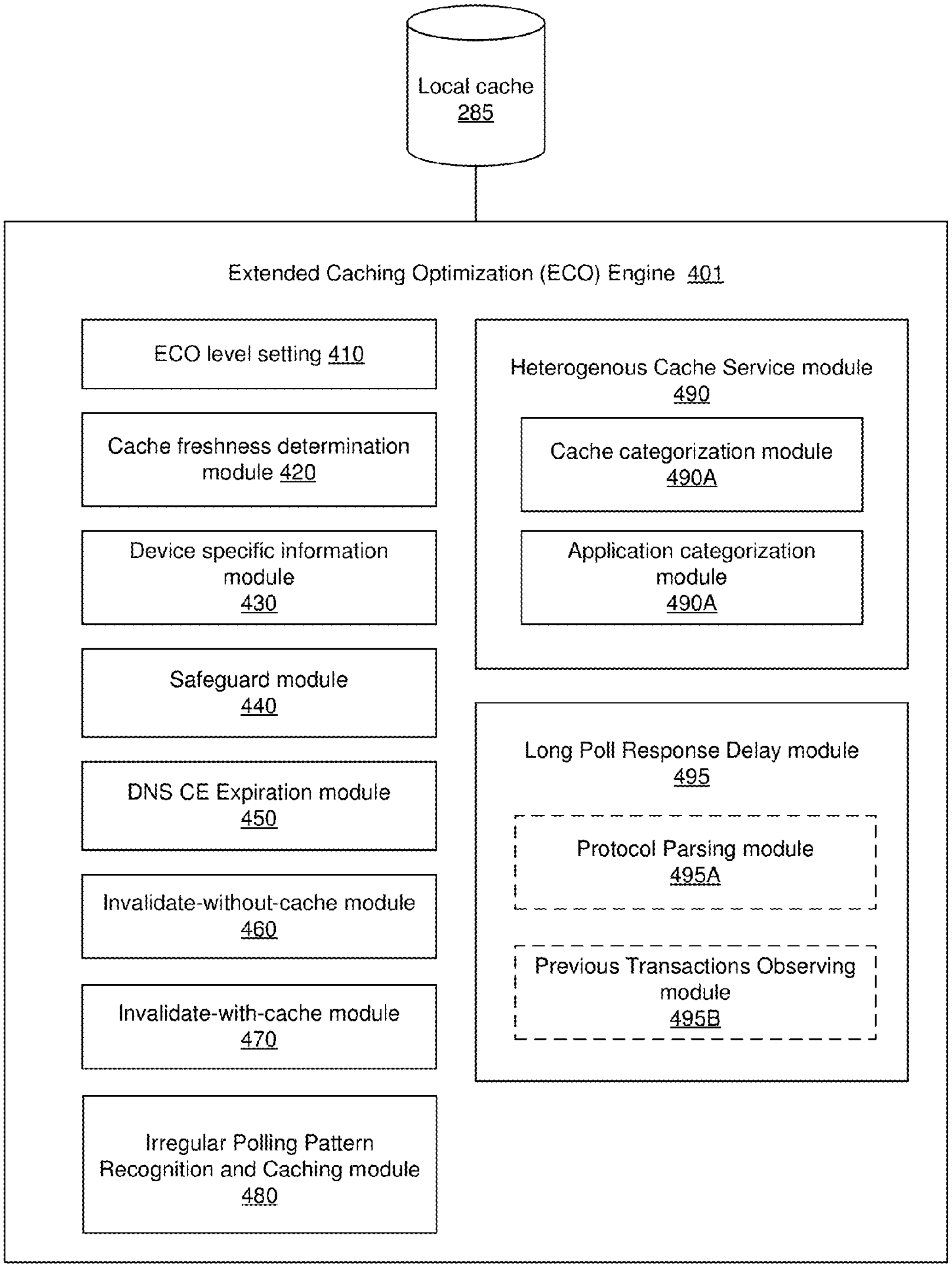

Local cache 285
Extended Caching Optimization (ECO) Engine 401
ECO level setting 410
Cache freshness determination module 420
Device specific information module 430
Safeguard module 440
DNS CE Expiration module 450
Invalidate-without-cache module 460
Invalidate-with-cache module 470
Irregular Polling Pattern Recognition and Caching module 480
Heterogenous Cache Service module 490
Cache categorization module 490A
Application categorization module 490A
Long Poll Response Delay module 495
Protocol Parsing module 495A
Previous Transactions Observing module 495B

*FIG. 4B*

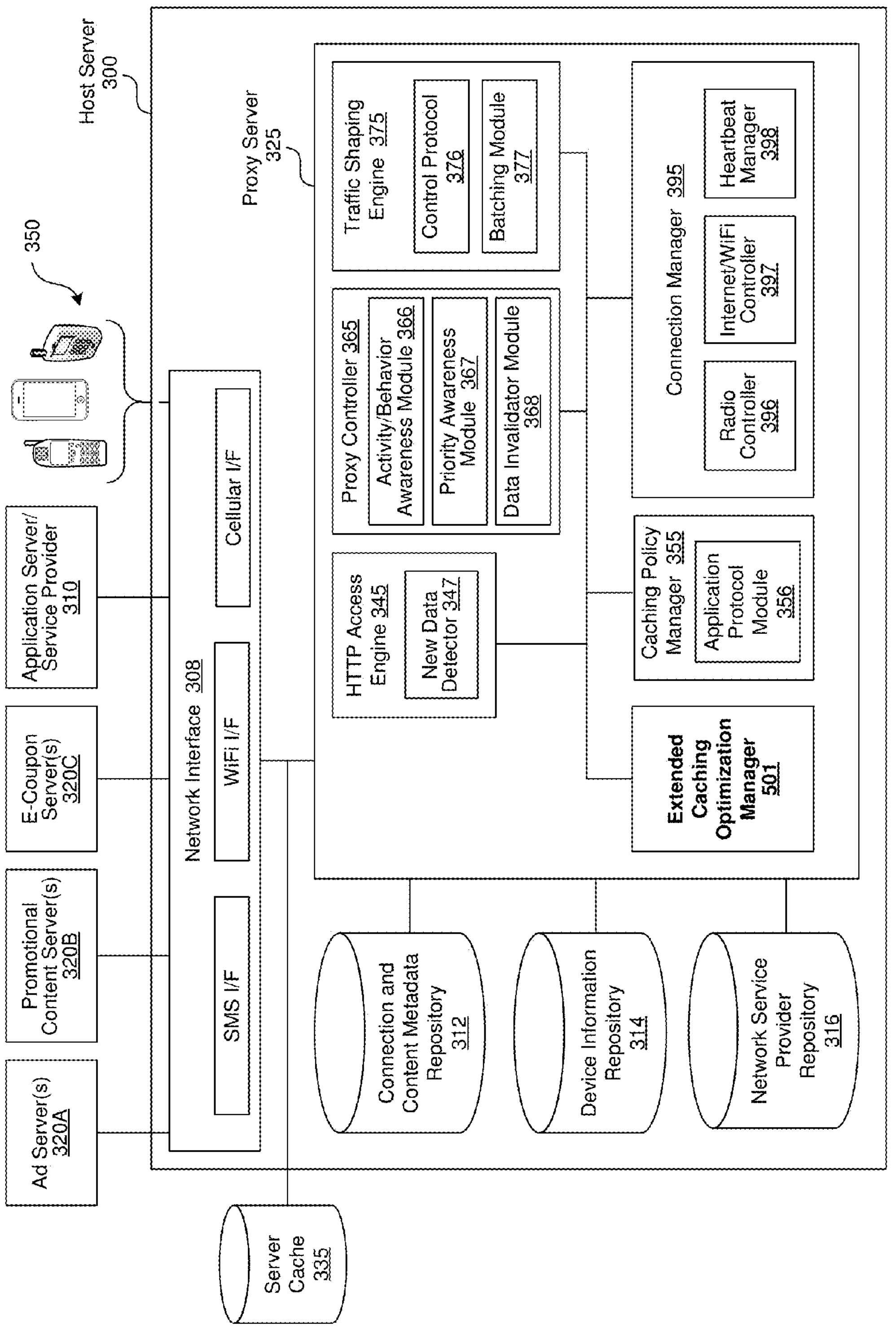

Host Server 300
350
Proxy Server 325
Application Server/ Service Provider 310
E-Coupon Server(s) 320C
Promotional Content Server(s) 320B
Ad Server(s) 320A
Network Interface 308
Cellular I/F
WiFi I/F
SMS I/F
Traffic Shaping Engine 375
Control Protocol 376
Batching Module 377
Proxy Controller 365
Activity/Behavior Awareness Module 366
Priority Awareness Module 367
Data Invalidator Module 368
Connection Manager 395
Radio Controller 396
Internet/WiFi Controller 397
Heartbeat Manager 398
HTTP Access Engine 345
New Data Detector 347
Caching Policy Manager 355
Application Protocol Module 356
Extended Caching Optimization Manager 501
Connection and Content Metadata Repository 312
Device Information Repository 314
Network Service Provider Repository 316
Server Cache 335

FIG. 5A

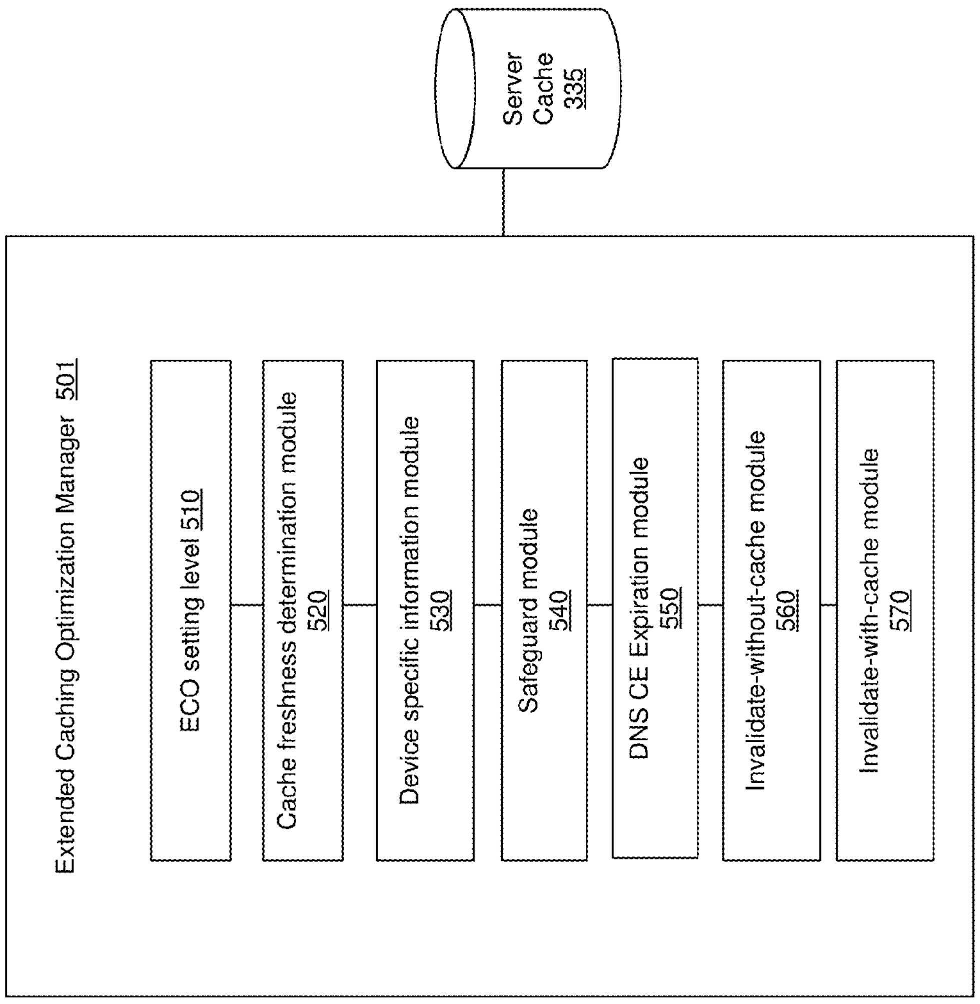

Server Cache 335
Extended Caching Optimization Manager 501
ECO setting level 510
Cache freshness determination module 520
Device specific information module 530
Safeguard module 540
DNS CE Expiration module 550
Invalidate-without-cache module 560
Invalidate-with-cache module 570

FIG. 5B

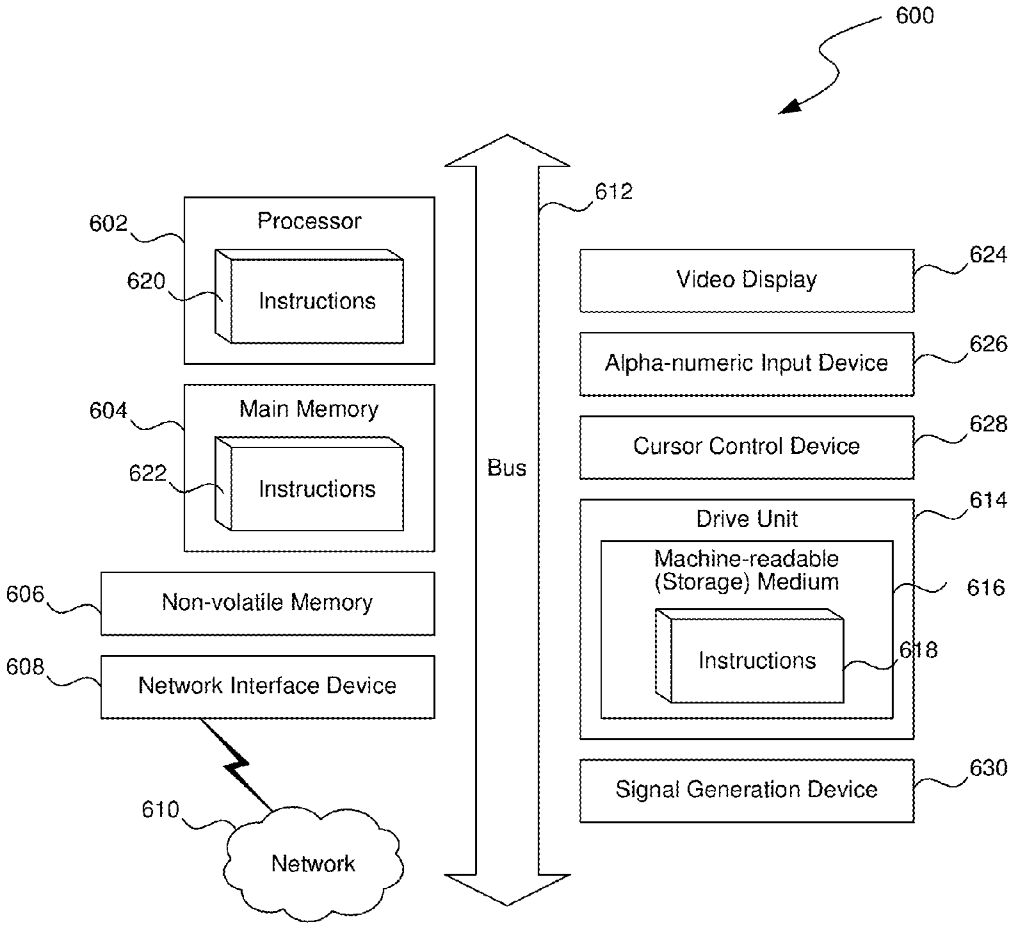

600
612
Bus
602
Processor
620
Instructions
604
Main Memory
622
Instructions
606
Non-volatile Memory
608
Network Interface Device
610
Network
624
Video Display
626
Alpha-numeric Input Device
628
Cursor Control Device
614
Drive Unit
Machine-readable (Storage) Medium
616
Instructions
618
630
Signal Generation Device

FIG. 6

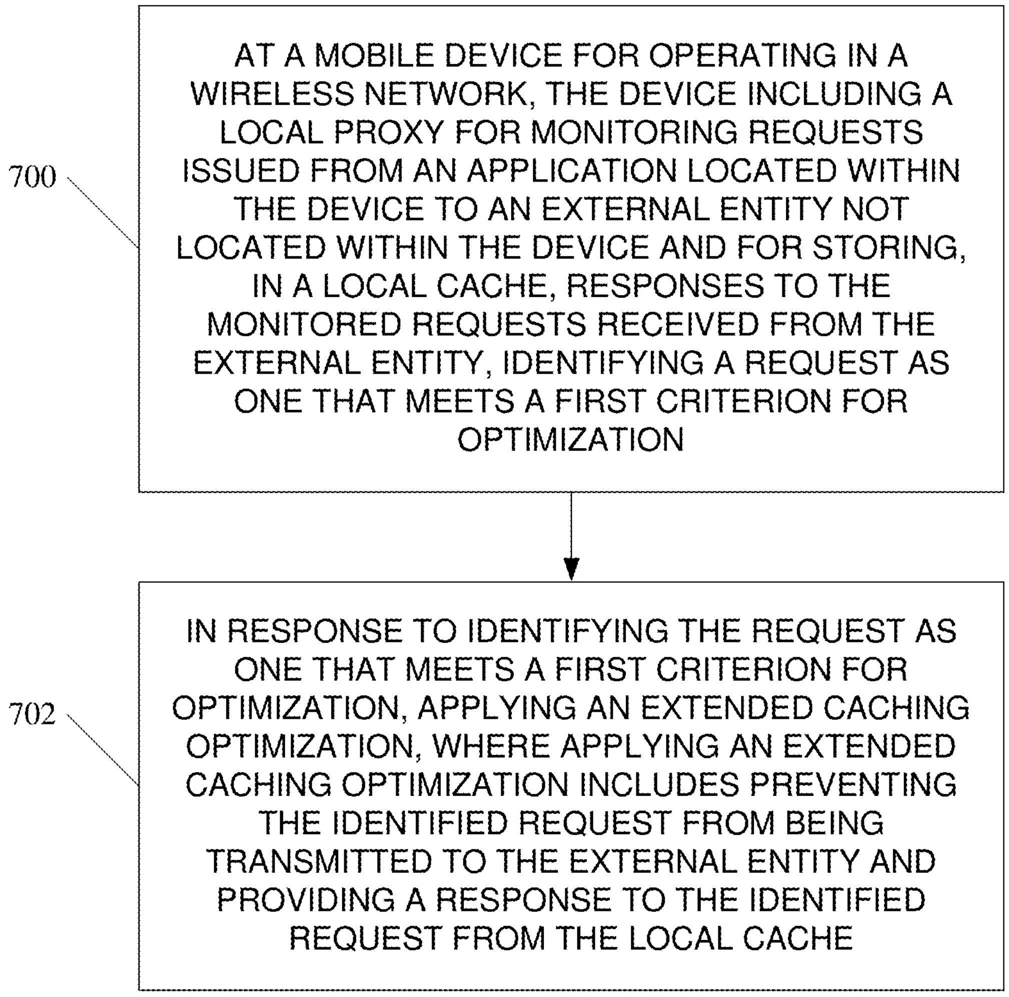

700
AT A MOBILE DEVICE FOR OPERATING IN A WIRELESS NETWORK, THE DEVICE INCLUDING A LOCAL PROXY FOR MONITORING REQUESTS ISSUED FROM AN APPLICATION LOCATED WITHIN THE DEVICE TO AN EXTERNAL ENTITY NOT LOCATED WITHIN THE DEVICE AND FOR STORING, IN A LOCAL CACHE, RESPONSES TO THE MONITORED REQUESTS RECEIVED FROM THE EXTERNAL ENTITY, IDENTIFYING A REQUEST AS ONE THAT MEETS A FIRST CRITERION FOR OPTIMIZATION
702
IN RESPONSE TO IDENTIFYING THE REQUEST AS ONE THAT MEETS A FIRST CRITERION FOR OPTIMIZATION, APPLYING AN EXTENDED CACHING OPTIMIZATION, WHERE APPLYING AN EXTENDED CACHING OPTIMIZATION INCLUDES PREVENTING THE IDENTIFIED REQUEST FROM BEING TRANSMITTED TO THE EXTERNAL ENTITY AND PROVIDING A RESPONSE TO THE IDENTIFIED REQUEST FROM THE LOCAL CACHE

*FIG. 7*

# DISTRIBUTED CACHING SYSTEMS WITH CONFIGURABLE EXTENDED CACHING OPTIMIZATION

## PRIORITY CLAIM

This application is a continuation of International Patent Application Serial No. PCT/US14/46537, filed Jul. 14, 2014, which claims the benefit of U.S. Provisional Patent Application Ser. No. 61/845,752, filed Jul. 12, 2013; U.S. Provisional Patent Application Ser. No. 61/856,343, filed Jul. 19, 2013; U.S. Provisional Patent Application Ser. No. 61/857,114, filed Jul. 22, 2013; U.S. Provisional Patent Application Ser. No. 61/858,013, filed Jul. 24, 2013; U.S. Provisional Patent Application Ser. No. 61/859,056, filed Jul. 26, 2013; U.S. Provisional Patent Application Ser. No. 61/859,364, filed Jul. 29, 2013: U.S. Provisional Patent Application Ser. No. 61/860,331, filed Jul. 31, 2013; and U.S. Provisional Patent Application Ser. No. 61/861,933, filed Aug. 2, 2013, the disclosures of each of which are incorporated herein by reference in their entireties.

## TECHNICAL FIELD

This disclosure relates to signaling optimization in telecommunication networks and data networks. More specifically, it relates to methods and systems for reducing traffic to and from a mobile device via the implementation of distributed caching systems with configurable extended caching optimization.

## BACKGROUND

The constant connections and disconnections of a mobile device to services and entities within a telecommunication and/or data network increase the amount of signaling network traffic within that network, which lowers the performance of the network overall. This imposes a burden upon network operators that forces them to increase bandwidth and network access.

It is very common for a mobile device to be receiving data from multiple sources (e.g., servers, web-sites, nodes of a network, etc.) in the service network. Smart phones, for example, may run several applications in parallel, and each application may engage in a periodic or non-periodic handshake with a network server, such as to check to see if there is any content to be downloaded to the mobile device or uploaded from the mobile device, to determine whether the connection between the mobile device and server should still be maintained, and so on.

In some circumstances, these handshaking interactions have little or no value to a user, such as when the user is not currently engaged in active use of the device, e.g., when the screen is dark or when the device is lying unused in a purse or backpack. Even when the user is actively using the device, an application on the device that continually polls a server to determine if there is content, such as text messages, email, etc., available may generate needless traffic if there is no content for the mobile device/user of that device.

In addition, such transactions typically put the mobile device radio in a high-power mode for a considerable length of time—typically between 15-30 seconds. As the high-power mode can consume as much as 100× the power as an idle mode, these network-initiated applications are power hungry and can quickly drain the battery. The issue has been exacerbated by the rapid increase of the popularity of applications with network-initiated functionalities, such as

push email, news feeds, status updates, multimedia content sharing and other mobile applications, etc. Furthermore, the problem with constant polling is that mobile phones also rely on signaling to send and receive calls and SMS messages and sometimes these basic mobile functions are forced to take a backseat to unruly applications and other mobile clients.

Thus, not only do these transactions consume battery power of the device—e.g., to activate an otherwise dormant radio transceiver circuit—the traffic so generated by these handshaking transactions consumes wireless bandwidth, such as between the mobile device and the cell tower, for example.

Therefore, in light of these disadvantages associated with conventional interactions between applications residing on a mobile device and the network entities with which the mobile device and its applications may interact, there is a need to optimize or reduce this kind of traffic. More specifically, there is a need for distributed caching systems with configurable extended caching optimization.

## SUMMARY

According to one aspect, a method for distributed caching of information using extended caching optimization. The method includes, at a mobile device for operating in a wireless network, monitoring requests issued from an application located within the device to an external entity not located within the device; storing, in a local cache, responses to the monitored requests received from the external entity; and, in response to identifying a request as one that meets a first criterion for optimization, applying an extended caching optimization, including preventing the identified request from being transmitted to the external entity and providing a response to the identified request from the local cache.

According to another aspect, the subject matter described herein includes a system for distributed caching of information using extended caching optimization. The system includes a mobile device for operating in a wireless network. The device includes a local cache and a local proxy for monitoring requests issued from an application located within the device to an external entity not located within the device and for storing, in the local cache, responses to the monitored requests received from the external entity. The local proxy identifies a request as one that meets a first criterion for optimization and applys an extended caching optimization, including preventing the identified request from being transmitted to the external entity and providing a response to the identified request from the local cache.

According to yet another aspect, the subject matter described herein includes a computer program product for distributed caching of information using extended caching optimization. The computer program product includes a non-transitory computer readable storage medium having computer readable code embodied therewith, the computer readable code configured for at a mobile device for operating in a wireless network, monitoring requests issued from an application located within the device to an external entity not located within the device; storing, in a local cache, responses to the monitored requests received from the external entity; and, in response to identifying a request as one that meets a first criterion for optimization, applying an extended caching optimization, including preventing the identified request from being transmitted to the external entity and providing a response to the identified request from the local cache.

## BRIEF DESCRIPTION OF THE DRAWINGS

The present embodiments are illustrated by way of example and are not intended to be limited by the figures of the accompanying drawings. In the drawings:

FIG. **1**A depicts a diagram illustrating example resources, including an extended caching optimization engine and an extended caching optimization manager that can function individually and/or together to implement the techniques disclosed herein;

FIG. **1**B depicts an example diagram of a system where a host server facilitates management of traffic, content caching, and/or resource conservation, and/or extended caching optimization;

FIG. **1**C depicts an example diagram of a proxy and cache system distributed between the host server and device which facilitates network traffic management and/or extended caching optimization;

FIG. **1**D depicts an example diagram of the logical architecture of a distributed proxy and cache system;

FIG. **1**E depicts an example diagram showing the architecture of client side components in a distributed proxy and cache system with an extended caching optimization engine implemented on the client-side proxy;

FIG. **1**F depicts an example diagram of the example components on the server side of the distributed proxy and cache system with an extended caching optimization manager implemented on the server-side proxy;

FIG. **1**G depicts an example diagram of a signaling optimizer of the distributed proxy and cache system;

FIG. **1**H depicts an example diagram of an example client-server architecture of the distributed proxy and cache system;

FIG. **1**I depicts an example diagram illustrating data flows between example client side components in a distributed proxy and cache system;

FIG. **2**A depicts a block diagram illustrating an example of client-side components in a distributed proxy and cache system residing on a mobile device (e.g., wireless device) that manages traffic in a wireless network (or broadband network) for resource conservation, content caching, traffic management, and/or extended caching optimization;

FIG. **2**B depicts a block diagram illustrating a further example of components in the cache system shown in the example of FIG. **2**A;

FIG. **2**C depicts a block diagram illustrating additional components in the application behavior detector and the caching policy manager in the cache system shown in the example of FIG. **2**A;

FIG. **2**D depicts a block diagram illustrating examples of additional components in the local cache shown in the example of FIG. **2**A;

FIG. **3**A depicts a block diagram illustrating an example of server-side components in a distributed proxy and cache system that manages traffic in a wireless network (or broadband network) for resource conservation, content caching, traffic management, and/or extended caching optimization;

FIG. **3**B depicts a block diagram illustrating a further example of components in the caching policy manager in the cache system shown in the example of FIG. **3**A;

FIG. **3**C depicts a block diagram illustrating another example of components in the proxy system shown in the example of FIG. **3**A;

FIG. **3**D depicts a block diagram illustrating examples of additional components in proxy server shown in the example of FIG. **3**A;

FIG. **4**A depicts a block diagram illustrating another example of client-side components in a distributed proxy and cache system, further including an extended caching optimization engine;

FIG. **4**B depicts a block diagram illustrating additional components in the extended caching optimization engine shown in the example of FIG. **4**A;

FIG. **5**A depicts a block diagram illustrating an example of server-side components in a distributed proxy and cache system, further including an extended caching optimization manager;

FIG. **5**B depicts a block diagram illustrating additional components in the extended caching optimization manager shown in the example of FIG. **5**A;

FIG. **6** depicts a diagrammatic representation of a machine in the example form of a computer system within which a set of instructions, for causing the machine to perform any one or more of the methodologies discussed herein, may be executed; and

FIG. **7** depicts a flowchart illustrating an exemplary process for distributed caching of information using extended caching optimization according to an embodiment of the subject matter described herein.

The same reference numbers and any acronyms identify elements or acts with the same or similar structure or functionality throughout the drawings and specification for ease of understanding and convenience.

## DETAILED DESCRIPTION

The following description and drawings are illustrative and are not to be construed as limiting. Numerous specific details are described to provide a thorough understanding of the disclosure. However, in certain instances, well-known or conventional details are not described in order to avoid obscuring the description. References to one or an embodiment in the present disclosure can be, but not necessarily are, references to the same embodiment; and, such references mean at least one of the embodiments.

Reference in this specification to "one embodiment" or "an embodiment" means that a particular feature, structure, or characteristic described in connection with the embodiment is included in at least one embodiment of the disclosure. The appearances of the phrase "in one embodiment" in various places in the specification are not necessarily all referring to the same embodiment, nor are separate or alternative embodiments mutually exclusive of other embodiments. Moreover, various features are described which may be exhibited by some embodiments and not by others. Similarly, various requirements are described which may be requirements for some embodiments but not other embodiments.

The terms used in this specification generally have their ordinary meanings in the art, within the context of the disclosure, and in the specific context where each term is used. Certain terms that are used to describe the disclosure are discussed below, or elsewhere in the specification, to provide additional guidance to the practitioner regarding the description of the disclosure. For convenience, certain terms may be highlighted, for example using italics and/or quotation marks. The use of highlighting has no influence on the scope and meaning of a term; the scope and meaning of a term is the same, in the same context, whether or not it is highlighted. It will be appreciated that same thing can be said in more than one way.

Consequently, alternative language and synonyms may be used for any one or more of the terms discussed herein, nor is any special significance to be placed upon whether or not a term is elaborated or discussed herein. Synonyms for certain terms are provided. A recital of one or more synonyms does not exclude the use of other synonyms. The use of examples anywhere in this specification including examples of any terms discussed herein is illustrative only, and is not intended to further limit the scope and meaning of the disclosure or of any exemplified term. Likewise, the disclosure is not limited to various embodiments given in this specification.

Without intent to limit the scope of the disclosure, examples of instruments, apparatus, methods and their related results according to the embodiments of the present disclosure are given below. Note that titles or subtitles may be used in the examples for convenience of a reader, which in no way should limit the scope of the disclosure. Unless otherwise defined, all technical and scientific terms used herein have the same meaning as commonly understood by one of ordinary skill in the art to which this disclosure pertains. In the case of conflict, the present document, including definitions will control.

FIG. **1**A depicts a diagram **10** illustrating example resources that implement the extended caching optimization techniques disclosed herein. Included in the diagram **10** are a mobile device **101**, a host server **111**, a plurality of third-party servers **119**, and a communications network **117**.

The mobile device **101** and host servers **111**, **119** are coupled in communication for data transmission over the network **117**. For example, the components may be connected via a twisted pair cabling network, a coax cable network, a telephonic network, or any suitable type of connection network. In some embodiments, the network **117** may be wireless. The technologies supporting the communications between the mobile device **101** and host servers **111**, **119** may include Ethernet, cellular, WiFi, and/or other suitable types of area network technologies. One of ordinary skill in the art will understand that the components of FIG. **1** are just one implementation of the computer network environment within which present embodiments may be implemented, and the various alternative embodiments are within the scope of the present embodiments. For example, the network **117** may include intervening devices (e.g., switches, routers, hubs, base stations, etc.) in the network **117**. In some examples, the network **117** comprises the Internet. Depending on the embodiments, mobile device **101** can be connected directly to the host server **111**, or via the network **117**, or both.

The host server **111** may be one or more server computers or work stations that are employed by a merchant for hosting websites that function as a channel to customer users for browsing products and placing purchase orders. The host server **111** typically includes at least one processor and a memory, and may be further connected to one or more computers (not shown in FIG. **1** for simplicity) that manage inventory, logistics and/or other commercial functions via the network **117**. The host server **111** may be a host server that facilitates management of traffic, content caching, and/or resource conservation (e.g., the host server **100**, described in FIG. **1**B below) or another server that is separate from the host server **100**. Depending on the embodiments, this separate server may be a portion of the host server **100**, or it may be hosted by a third party (e.g., the third-party server **119**).

The mobile device **101**, which may be used by a customer user to communicate with the host server **111**, may include a laptop, a tablet, a personal computer, a personal digital assistant ("PDA"), a smart phone, and the like. The mobile device **101** typically includes a display (not shown in FIG. **1** for simplicity), and may include suitable input devices (not shown for simplicity) such as a keyboard, a mouse, or a touchpad. In some embodiments, the display may be a touch-sensitive screen that includes input functionalities.

Applications **102** (e.g., applications **102**A-**102**N) are example applications of the mobile device **101**. Applications **102** on mobile device **101** can communicate directly to the third-party servers **119** via the network **117**. Some examples of applications **102** include news application, weather services, email clients, and/or social network applications. In general, each application **102** has a plurality of data relevant or necessary to the normal operations of the application. It is also typical that these applications **102** routinely communicate with the third-party servers **119** (e.g., via well-known polling techniques) for any update, and receive the updates via the network **117**, using one or more radio communication modules (not shown in FIG. **1** for simplicity).

Signal Optimization

The embodiments disclosed herein recognize that the number and capability of mobile devices have increased dramatically in recent years, putting tremendous pressure on mobile carriers to optimize and manage finite network resources. Subscribers use their devices more intensively than ever, installing any number of countless thousands of applications. Carriers have no control over what applications are installed on end-user devices or the behavior of those apps. This has led to a loss of operator control over mobile data traffic itself.

Accordingly, as is described in more detail below, the present disclosure addresses these challenges by optimizing and managing signaling activity, increasing control over data traffic for mobile carriers. It conserves network bandwidth, freeing capacity for uses that provide the highest value to subscribers.

More specifically, many mobile applications (e.g., applications **102**A-**102**N) regularly poll their application servers **119**A-**119**N to check for new data. Every time an application (e.g., application **102**A) checks for updates, even when no new data is available, the device signals the wireless network (e.g., network **117**). This signaling activity creates congestion as the radio network is overwhelmed with constant requests to connect. Network bandwidth is wasted when applications repeatedly download unchanged content. Carriers face increasing costs as they are forced to expand capacity to accommodate both excessive signaling activity and increasing demand for bandwidth. Unnecessary mobile signaling has an adverse effect on end-user experience, including longer setup times, slower speeds, and even denial of service. It results in shorter battery life for devices (e.g., device **101**)—an issue that impacts both subscribers and device manufacturers.

In general, the signaling optimization techniques disclosed herein significantly reduce mobile signaling to relieve mobile network congestion. The techniques can manage the exchange of control information and content between mobile devices and the network, using various virtualized proxy and caching technologies. They can analyze mobile application data requests, transparently detect redundant traffic patterns and cache the results of unnecessary requests on the client. The signal optimization server (e.g., host server **111**) polls for updates, so that the client (e.g., device **101**, through local proxy **105**) connects to the network only when updates are available. Optimizing signaling at the handset prevents consumption of network resources, enabling more efficient use of those resources. Caching content on the client (e.g., at local cache **185**) also reduces network bandwidth utilization. In one embodiment, local cache **185** may be used to store copies of requests received from applications **102**, copies of responses received from host server **111**, or other useful information.

The disclosed signaling optimization techniques reduce operators' costs by delaying the need for expensive upgrades to wireless network infrastructure. They improve subscriber experience by increasing service levels and extending the battery life of mobile devices.

Moreover, it is noted that the signaling optimization techniques disclosed herein are transparent to end users and to the operation of their mobile apps, with a minimal impact on device CPU and memory. The techniques which can be implemented in form of, for example, software require no (or little, if any) changes to applications or to the network itself, thereby supporting any underlying mobile network technology. These techniques can also complement other techniques for mobile traffic management, including compression and deep packet inspection.

Overall, among other benefits, the disclosed techniques can reduce wireless operator costs, conserve mobile network bandwidth, significantly reduce mobile signaling, delay the needs for wireless infrastructure upgrades, and extend device battery life and improves service levels.

In accordance with some embodiments, a client component (e.g., local proxy **105**) and a server component (e.g., proxy server **113**) can either individually or together or both perform the signal optimization techniques. Customers can choose between two deployment models for the server software. In some embodiments, Hosted deployment allows customers to get the solution up and running as quickly as possible.

In-network deployment of the server software is recommended for customers who want in-house, hands-on control over all aspects of the solution. In this model, the management server resides in the customer data center. Pushing optimization to the client stops unnecessary signaling before it can consume network resources. This is in contrast to conventional approaches to addressing the signaling challenge, which rely on network-side capabilities such as deep packet inspection.

Signal Optimization—Extended Caching

It is a goal for the network carriers seek to maximize the aggregate value provided by finite network resources across the entire subscriber base, which requires that these resources be allocated to their highest-value uses. This approach allows the carriers to maximize their ability to monetize infrastructure investments. The "Extended Caching" techniques, which are disclosed herein and will be described in more details below, are a powerful tool in achieving these goals.

More specifically, it is recognized by the disclosed embodiments that there is an inherent trade-off in the frequency of updates for mobile apps and the consumption of network resources. When no update is available, the present embodiments recognize that connecting to the network serves no useful purpose and needlessly consumes network resources.

Furthermore, even when an update is available, the user may not derive much or any benefit from it as not all updates are created equal—end users derive more benefit from some updates than others. Updates that occur when a device is actively being used are more useful and valuable to the user than those that occur when the device is asleep. The cost, in terms of mobile network resource consumed in providing updates, may easily exceed the benefit; especially if the device is not being actively used. For example, when the screen is not lit and the radio is inactive, the user may have left the device in another room. In this common situation, no benefit is received, but network resources are consumed. Thus, network resources are being misallocated in the sense that they are being applied in a way that yields little or no subscriber value. The subscriber base as a whole is better served if those network resources are allocated to users who are actually using their devices at the time. Information on the device state can serve as a useful tool for carriers to determine the benefit that an update provides.

As such, in some embodiments, such as system **10** shown in FIG. **1**A, the local proxy **105** includes an extended caching optimization engine **107**, and the proxy server **113** includes an extended caching optimization manager **115** that can function individually and/or together to implement the techniques disclosed herein. In one embodiment, proxy server **113** may include a server cache **135**, which may be used to store requests received from mobile device **101**, responses received from servers **119**, or other useful information.

Configurable Settings in Extended Caching Optimization

The extended caching optimization engine **107** (and/or the extended caching optimization manager **115**) includes configurable cache settings to fine tune the way the product responds to changes in cached resources. For example, extended caching optimization engine **107** can apply to situations when the screen is not lit or the radio is not connected to the network. These device states correspond to situations where the user is likely to receive less benefit from an update, even when one is available. Extended cache settings delay delivery of updates in device states where the end user receives less benefit from getting an update at that particular moment in time.

The extended caching optimization engine **107** (and/or the extended caching optimization manager **115**) includes a configurable, "tunable" parameter because carriers require fine-grained control to allocate finite resources to the highest-value use case. The choice of which configuration settings to use can be a judgment call on the part of the carrier, and can be made in the context of each carrier network's unique subscriber base, usage patterns, and behaviors. As an example of the flexibility of the configurable settings, the lowest setting level (default) tunes the product to favor priority of end-user experience over congestion relief/battery life. This is in contrast to the highest setting level that favors congestion relief/battery life over priority of end-user experience.

The Extended Caching settings can be modified in real time using either an application programming interface (API) (e.g., a proprietary "REST" API as provided by SEVEN Networks Inc.) or a Management Web Interface. Changes to settings are communicated to in-market devices immediately (e.g., from the optimization manager **115** to the optimization engine **107**). Devices (e.g., device **101**) that may be powered off at the time of the change receives the most recent update when they return to the network **117**.

In some embodiments, Extended Caching settings apply to all applications that are being optimized by Open Channel Signaling Optimization. In some other embodiments, Extended Caching to be enabled for selective applications.

Also, in one or more embodiments, Extended Caching levels can be set on an individual user basis. This can be done via integration with the REST API. In addition, some embodiments support creation of Extended Caching Groups which can be used to associate users with common data plans, etc., to specific Extended Caching settings.

These and various other embodiments and implementations of the disclosed extended caching optimization (ECO) techniques in achieving aggressive signal optimization, as well as various components in the embodiments of ECO techniques (e.g., ECO engine **107** and/or ECO manager **115**), are described in more details below. It is further noted that some specific examples of the extended caching optimization engine **107** and the extended caching optimization manager **115**, including their behaviors under different configurable settings in performing different functions (e.g., which may be aimed at solving different scenarios) are introduced in relation to FIGS. **4**A, **4**B, **5**A and **5**B.

FIG. **1**B illustrates an example diagram of a system where a host server **100** facilitates management of traffic, content caching, and/or resource conservation and/or signal optimization or extended caching optimization between mobile devices (e.g., wireless devices **150**), and an application server or content provider **110**, or other servers such as an ad server **120**A, promotional content server **120**B, or an e-coupon server **120**C in a wireless network (or broadband network) for resource conservation. The host server **100** can further interact with mobile or client devices **150** for getting reports and/or updates on resource usage, savings, and the like.

Client device **150** can be any system and/or device, and/or any combination of devices/systems that is able to establish a connection, including wired, wireless, and cellular connections, with another device, a server, and/or other systems, such as host server **100** and/or application server/content provider **110**. Client devices **150** may provide to a user **103** a user interface **104**, which may include a display and/or other output functionalities to present information and data exchanged between among the devices **150** and/or the host server **100** and/or application server/content provider **110**. The application server/content provider **110** can by any server, including third party servers or service/content providers further including advertisement, promotional content, publication, or electronic coupon servers or services. Similarly, separate advertisement servers **120**A, promotional content servers **120**B, and/or e-Coupon servers **120**C as application servers or content providers are illustrated by way of example.

For example, client devices **150** can include mobile, hand held or portable devices, wireless devices, or non-portable devices and can be any of, but not limited to, a server desktop, a desktop computer, a computer cluster, or portable devices, including a notebook, a laptop computer, a handheld computer, a palmtop computer, a mobile phone, a cell phone, a smart phone, a PDA, a Blackberry device, a Palm device, a handheld tablet (e.g., an iPad or any other tablet), a hand held console, a hand held gaming device or console, any SuperPhone such as the iPhone, and/or any other portable, mobile, hand held devices, or fixed wireless interface such as a M2M device, etc. In one embodiment, the client devices **150**, host server **100**, and application server **110** are coupled via a network **106** and/or a network **108**. In one embodiment, network **106** may be a cellular or mobile network, which device **150** may connect to via a base station, radio network controller, or radio access network, represented in FIG. **1**B as a cell phone tower **112**. In some embodiments, the devices **150** and host server **100** may be directly connected to one another.

The input mechanism on client devices **150** can include touch screen keypad (including single touch, multi-touch, gesture sensing in 2D or 3D, etc.), a physical keypad, a mouse, a pointer, a track pad, motion detector (e.g., including 1-axis, 2-axis, 3-axis accelerometer, etc.), a light sensor, capacitance sensor, resistance sensor, temperature sensor, proximity sensor, a piezoelectric device, device orientation

detector (e.g., electronic compass, tilt sensor, rotation sensor, gyroscope, accelerometer), or a combination of the above.

Signals received or detected indicating user activity at client devices **150** through one or more of the above input mechanism, or others, can be used in the disclosed technology in acquiring context awareness at the client device **150**. Context awareness at client devices **150** generally includes, by way of example but not limitation, client device **150** operation or state acknowledgement, management, user activity/behavior/interaction awareness, detection, sensing, tracking, trending, and/or application (e.g., mobile applications) type, behavior, activity, operating state, etc.

Context awareness in the present disclosure also includes knowledge and detection of network side contextual data and can include network information such as network capacity, bandwidth, traffic, type of network/connectivity, and/or any other operational state data and/or mobile application loading and/or activities. Network side contextual data can be received from and/or queried from network service providers (e.g., cell provider **112** and/or Internet service providers) of the network **106** and/or network **108** (e.g., by the host server and/or devices **150**). In addition to application context awareness as determined from the client **150** side, the application context awareness may also be received from or obtained/queried from the respective application/service providers **110** (by the host **100** and/or client devices **150**).

The host server **100** can use, for example, contextual information obtained for client devices **150**, networks **106**/**108**, applications (e.g., mobile applications), application server/provider **110**, or any combination of the above, to manage the traffic in the system to satisfy data needs of the client devices **150** (e.g., to satisfy application or any other request including HTTP request). In one embodiment, the traffic is managed by the host server **100** to satisfy data requests made in response to explicit or non-explicit user **103** requests and/or device/application maintenance tasks. The traffic can be managed such that network consumption, for example, use of the cellular network is conserved for effective and efficient bandwidth utilization. In addition, the host server **100** can manage and coordinate such traffic in the system such that use of device **150** side resources (e.g., including but not limited to battery power consumption, radio use, processor/memory use) are optimized with a general philosophy for resource conservation while still optimizing performance and user experience.

For example, in context of battery conservation, the device **150** can observe user activity (for example, by observing user keystrokes, backlight status, or other signals via one or more input mechanisms, etc.) and alter device **150** behaviors. The device **150** can also request the host server **100** to alter the behavior for network resource consumption based on user activity or behavior.

In one embodiment, the traffic management for resource conservation and/or mobile application offloading are performed using a distributed system between the host server **100** and client device **150**. The distributed system can include proxy server and cache components on the server side **100** and on the device/client side, for example, as shown by the server cache **135** on the server **100** side and the local cache **185** on the client **150** side.

Functions and techniques disclosed for context aware traffic management and/or mobile application offloading for resource conservation in networks (e.g., network **106** and/or **108**) and devices **150**, can reside in a distributed proxy and cache system. The proxy and cache system can be distributed between, and reside on, a given client device **150** in part or in whole and/or host server **100** in part or in whole. The distributed proxy and cache system are illustrated with further reference to the example diagram shown in FIG. **1**C. Notably, in some embodiments of such systems, the host server **100** can include or correspond to the host server **111** (FIG. **1**A), the application server **110** can include or correspond to the third-party servers **119** (FIG. **1**A), and/or the mobile device **150** can include or correspond to the mobile device **101** (FIG. **1**A).

In one embodiment, client devices **150** communicate with the host server **100** and/or the application server **110** over network **106**, which can be a cellular network and/or a broadband network. To facilitate overall traffic management and/or signal optimization between devices **150** and various application servers/content providers **110** to implement network (bandwidth utilization) and device resource (e.g., battery consumption), the host server **100** can communicate with the application server/providers **110** over the network **108**, which can include the Internet (e.g., a broadband network).

In general, the networks **106** and/or **108**, over which the client devices **150**, the host server **100**, and/or application server **110** communicate, may be a cellular network, a broadband network, a telephonic network, an open network, such as the Internet, or a private network, such as an intranet and/or the extranet, or any combination thereof. For example, the Internet can provide file transfer, remote log in, email, news, RSS, cloud-based services, instant messaging, visual voicemail, push mail, VoIP, and other services through any known or convenient protocol, such as, but is not limited to the TCP/IP protocol, UDP, HTTP, DNS, FTP, UPnP, NSF, ISDN, PDH, RS-232, SDH, SONET, etc.

The networks **106** and/or **108** can be any collection of distinct networks operating wholly or partially in conjunction to provide connectivity to the client devices **150** and the host server **100** and may appear as one or more networks to the serviced systems and devices. In one embodiment, communications to and from the client devices **150** can be achieved by, an open network, such as the Internet, or a private network, broadband network, such as an intranet and/or the extranet. In one embodiment, communications can be achieved by a secure communications protocol, such as secure sockets layer (SSL), or transport layer security (TLS).

In addition, communications can be achieved via one or more networks, such as, but are not limited to, one or more of WiMax, a Local Area Network (LAN), Wireless Local Area Network (WLAN), a Personal area network (PAN), a Campus area network (CAN), a Metropolitan area network (MAN), a Wide area network (WAN), a Wireless wide area network (WWAN), or any broadband network, and further enabled with technologies such as, by way of example, Global System for Mobile Communications (GSM), Personal Communications Service (PCS), Bluetooth, WiFi, Fixed Wireless Data, 2G, 2.5G, 3G, 4G, IMT-Advanced, pre-4G, LTE Advanced, mobile WiMax, WiMax 2, WirelessMAN-Advanced networks, enhanced data rates for GSM evolution (EDGE), General packet radio service (GPRS), enhanced GPRS, iBurst, UMTS, HSPDA, HSUPA, HSPA, UMTS-TDD, 1xRTT, EV-DO, messaging protocols such as, TCP/IP, SMS, MMS, extensible messaging and presence protocol (XMPP), real time messaging protocol (RTMP), instant messaging and presence protocol (IMPP), instant messaging, USSD, IRC, or any other wireless data networks, broadband networks, or messaging protocols.

FIG. **1**C illustrates an example diagram of a proxy and cache system distributed between the host server **100** and device **150** which facilitates network traffic management and/or signal optimization (including extended caching) between the device **150** and an application server or content provider **110**, or other servers such as an ad server **120**A, promotional content server **120**B, or an e-coupon server **120**C for resource conservation and content caching The proxy system distributed among the host server **100** and the device **150** can further track alarms, timers or other triggers implemented by applications on a device and resources used by such alarms, timers, or other triggers to determine associations using which the proxy system can manipulate the alarms, timers or other triggers to occur at an optimal time to reduce resource usage.

The distributed proxy and cache system can include, for example, the proxy server **125** (e.g., remote proxy) and the server cache, **135** components on the server side. The server-side proxy **125** and cache **135** can, as illustrated, reside internal to the host server **100**. In addition, the proxy server **125** and cache **135** on the server-side can be partially or wholly external to the host server **100** and in communication via one or more of the networks **106** and **108**. For example, the proxy server **125** may be external to the host server and the server cache **135** may be maintained at the host server **100**. Alternatively, the proxy server **125** may be within the host server **100** while the server cache is external to the host server **100**. In addition, each of the proxy server **125** and the cache **135** may be partially internal to the host server **100** and partially external to the host server **100**. The application server/content provider **110** can by any server including third party servers or service/content providers further including advertisement, promotional content, publication, or electronic coupon servers or services. Similarly, separate advertisement servers **120**A, promotional content servers **120**B, and/or e-Coupon servers **120**C as application servers or content providers are illustrated by way of example.

The distributed system can also, include, in one embodiment, client-side components, including by way of example but not limitation, a local proxy **175** (e.g., a mobile client on a mobile device) and/or a local cache **185**, which can, as illustrated, reside internal to the device **150** (e.g., a mobile device).

In addition, the client-side proxy **175** and local cache **185** can be partially or wholly external to the device **150** and in communication via one or more of the networks **106** and **108**. For example, the local proxy **175** may be external to the device **150** and the local cache **185** may be maintained at the device **150**. Alternatively, the local proxy **175** may be within the device **150** while the local cache **185** is external to the device **150**. In addition, each of the proxy **175** and the cache **185** may be partially internal to the host server **100** and partially external to the host server **100**.

In one embodiment, the distributed system can include an optional caching proxy server **199**. The caching proxy server **199** can be a component which is operated by the application server/content provider **110**, the host server **100**, or a network service provider **112**, and or any combination of the above to facilitate network traffic management for network and device resource conservation. Proxy server **199** can be used, for example, for caching content to be provided to the device **150**, for example, from one or more of, the application server/provider **110**, host server **100**, and/or a network service provider **112**. Content caching can also be entirely or partially performed by the remote proxy **125** to satisfy application requests or other data requests at the device **150**.

In context aware traffic management and optimization for resource conservation in a network (e.g., cellular or other wireless networks), characteristics of user activity/behavior and/or application behavior at a mobile device (e.g., any wireless device) **150** can be tracked by the local proxy **175** and communicated, over the network **106** to the proxy server **125** component in the host server **100**, for example, as connection metadata. The proxy server **125** which in turn is coupled to the application server/provider **110** provides content and data to satisfy requests made at the device **150**.

In addition, the local proxy **175** can identify and retrieve mobile device properties, including one or more of, battery level, network that the device is registered on, radio state, or whether the mobile device is being used (e.g., interacted with by a user). In some instances, the local proxy **175** can delay, expedite (prefetch), and/or modify data prior to transmission to the proxy server **125**, when appropriate.

The local database **185** can be included in the local proxy **175** or coupled to the local proxy **175** and can be queried for a locally stored response to the data request prior to the data request being forwarded on to the proxy server **125**. Locally cached responses can be used by the local proxy **175** to satisfy certain application requests of the mobile device **150**, by retrieving cached content stored in the cache storage **185**, when the cached content is still valid.

Similarly, the proxy server **125** of the host server **100** can also delay, expedite, or modify data from the local proxy prior to transmission to the content sources (e.g., the application server/content provider **110**). In addition, the proxy server **125** uses device properties and connection metadata to generate rules for satisfying request of applications on the mobile device **150**. The proxy server **125** can gather real time traffic information about requests of applications for later use in optimizing similar connections with the mobile device **150** or other mobile devices.

In general, the local proxy **175** and the proxy server **125** are transparent to the multiple applications executing on the mobile device. The local proxy **175** is generally transparent to the operating system or platform of the mobile device and may or may not be specific to device manufacturers. In some instances, the local proxy **175** is optionally customizable in part or in whole to be device specific. In some embodiments, the local proxy **175** may be bundled into a wireless model, a firewall, and/or a router.

In one embodiment, the host server **100** can in some instances, utilize the store and forward functions of a short message service center (SMSC) **114**, such as that provided by the network service provider, in communicating with the device **150** in achieving network traffic management. Note that **114** can also utilize any other type of alternative channel including USSD or other network control mechanisms. The host server **100** can forward content or HTTP responses to the SMSC **114** such that it is automatically forwarded to the device **150** if available, and for subsequent forwarding if the device **150** is not currently available.

In general, the disclosed distributed proxy and cache system allows optimization of network usage, for example, by serving requests from the local cache **185**, the local proxy **175** reduces the number of requests that need to be satisfied over the network **106**. Further, the local proxy **175** and the proxy server **125** may filter irrelevant data from the communicated data. In addition, the local proxy **175** and the proxy server **125** can also accumulate low priority data and send it in batches to avoid the protocol overhead of sending individual data fragments. The local proxy **175** and the proxy server **125** can also compress or transcode the traffic, reducing the amount of data sent over the network **106** and/or **108**. The signaling traffic in the network **106** and/or **108** can be reduced, as the networks are now used less often and the network traffic can be synchronized among individual applications.

With respect to the battery life of the mobile device **150**, by serving application or content requests from the local cache **185**, the local proxy **175** can reduce the number of times the radio module is powered up. The local proxy **175** and the proxy server **125** can work in conjunction to accumulate low priority data and send it in batches to reduce the number of times and/or amount of time when the radio is powered up. The local proxy **175** can synchronize the network use by performing the batched data transfer for all connections simultaneously.

FIG. **1D** illustrates an example diagram of the logical architecture of a distributed proxy and cache system.

The distributed system can include, for example the following components:

Client Side Proxy **175**: a component installed in the Smartphone, mobile device or wireless device **150** that interfaces with device's operating system, as well as with data services and applications installed in the device. The client side proxy **175** is typically compliant with and able to operate with standard or state of the art networking protocols.

The server side proxy **125** can include one or more servers that can interface with third party application servers (e.g., **199**), mobile operator's network (which can be proxy **199** or an additional server that is not illustrated) and/or the client side proxy **175**, either directly or via an intermediary element, such as SMSC **114**. In general, the server side proxy **125** can be compliant with and is generally able to operate with standard or state of the art networking protocols and/or specifications for interacting with mobile network elements and/or third party servers.

Reporting and Usage Analytics Server **174**: The Reporting and Usage Analytics system or component **174** can collect information from the client side **175** and/or the server side **125** and provides the necessary tools for producing reports and usage analytics can used for analyzing traffic and signaling data. Such analytics can be used by the proxy system in managing/reducing network traffic or by the network operator in monitoring their networks for possible improvements and enhancements. Note that the reporting and usage analytics system/component **174** as illustrated, may be a server separate from the server-side proxy **125**, or it may be a component of the server-side proxy **125**, residing partially or wholly therein.

Notably, in some embodiments of such systems, the client-side proxy **175** can include or correspond to the local proxy **105** (FIG. **1A**), and the server-side proxy **125** can include or correspond to the proxy server **113** (FIG. **1A**).

FIG. **1E** illustrates an example diagram showing the architecture of client side components in a distributed proxy and cache system. In the embodiment illustrated in FIG. **1E**, client side components include the mobile operating system **162** and mobile applications **163**, collectively referred to as the mobile OS and apps **165**.

Other client side components include client side proxy **175**, which can include software components or agents installed on the mobile device that enables traffic optimization and performs the related functionalities on the client side. Components of the client side proxy **175** can operate transparently for end users and applications **163**. The client side proxy **175** can be installed on mobile devices for optimization to take place, and it can effectuate changes on the data routes. Once data routing is modified, the client side proxy **175** can respond to application requests to service providers or host servers, in addition to or instead of letting those applications **163** access data network directly. In general, applications **163** on the mobile device do not notice that the client side proxy **175** is responding to their requests. Some example components of the client side proxy **175** are described as follows:

Device State Monitor **121**: The device state monitor **121** can be responsible for identifying several states and metrics in the device, such as network status, display status, battery level, etc. (referred to as radio/battery/display information **161**), such that the remaining components in the client side proxy **175** can operate and make decisions according to device state, acting in an optimal way in each state.

Extended Caching Optimization Engine **177**: Similar to what are mentioned with regard to the optimization engine **107** in FIG. **1**A, the optimization engine **177** can perform, either individually or in conjunction with Extended Caching Optimization Manager **179** (FIG. **1**F, discussed below), the signal optimization techniques disclosed herein. In the embodiment shown in FIG. **1**E, the optimization engine **177** is coupled to the device state monitor **121** to receive application activity, battery, network status, display or LCD status (e.g., backlit status) as well as user selection, an administrator's selection, and/or other suitable information in determining, for example, user inactivity, radio availability, prediction of user activity (e.g., based on historical patterns), and/or network health state (e.g., congestion). The optimization engine **177** can also communicate with the server-side proxy **125** (FIG. **1**F) for selectively caching one or more operational data (e.g., requests and/or responses) of applications **163** (e.g., applications **102**, FIG. **1**A) to and from the remote host server **111** in carrying out the extended caching techniques discussed in, for example, FIG. **1**A and FIGS. **4**A-**5**B.

Traffic Recognizer **122**: The traffic recognizer **122** analyzes all traffic between the wireless device applications **163** and their respective host servers in order to identify recurrent patterns. Supported transport protocols include, for example, DNS, HTTP and HTTPS, such that traffic through those ports is directed to the client side proxy **175**. While analyzing traffic, the client side proxy **175** can identify recurring polling patterns which can be candidates to be performed remotely by the server side proxy **125**, and send to the protocol optimizer **123**.

Protocol Optimizer **123**: The protocol optimizer **123** can implement the logic of serving recurrent request from the local cache **185** instead of allowing those request go over the network to the service provider/application host server. One is its tasks is to eliminate or minimize the need to send requests to the network, positively affecting network congestion and device battery life.

Local Cache **185**: The local cache **185** can store responses to recurrent requests, and can be used by the Protocol Optimizer **123** to send responses to the applications **163**.

Traffic Scheduler **124**: The traffic scheduler **124** can temporally move communications to optimize usage of device resources by unifying keep-alive signaling so that some or all of the different applications **163** can send keep-alive messages at the same time (traffic pipelining). Traffic scheduler **124** may also decide to delay transmission of data that is not relevant at a given time (for example, when the device is not actively used).

Policy Manager **120**: The policy manager **120** can store and enforce traffic optimization and reporting policies provisioned by a Policy Management Server (PMS). At the client side proxy **175** first start, traffic optimization and

reporting policies (policy profiles) that is to be enforced in a particular device can be provisioned by the Policy Management Server.

Watch Dog **127**: The watch dog **127** can monitor the client side proxy **175** operating availability. In case the client side proxy **175** is not working due to a failure or because it has been disabled, the watchdog **127** can reset DNS routing rules information and can restore original DNS settings for the device to continue working until the client side proxy **175** service is restored.

Reporting Agent **126**: The reporting agent **126** can gather information about the events taking place in the device and sends the information to the Reporting Server. Event details are stored temporarily in the device and transferred to reporting server only when the data channel state is active. If the client side proxy **175** doesn't send records within twenty-four hours, the reporting agent **126** may attempt to open the connection and send recorded entries or, in case there are no entries in storage, an empty reporting packet. All reporting settings are configured in the policy management server.

Push Client **128**: The push client **128** can be responsible for the traffic to between the server side proxy **125** and the client side proxy **175**. The push client **128** can send out service requests like content update requests and policy update requests, and receives updates to those requests from the server side proxy **125**. In addition, push client **128** can send data to a reporting server (e.g., the reporting and/or usage analytics system which may be internal to or external to the server side proxy **125**).

The proxy server **199** has a wide variety of uses, from speeding up a web server by caching repeated requests, to caching web, DNS and other network lookups for a group of clients sharing network resources. The proxy server **199** is optional. The distributed proxy and cache system (**125** and/or **175**) allows for a flexible proxy configuration using either the proxy **199**, additional proxy(s) in operator's network, or integrating both proxies **199** and an operator's or other third-party's proxy.

FIG. **1**F illustrates a diagram of the example components on the server side of the distributed proxy and cache system.

The server side **125** of the distributed system can include, for example a relay server **142**, which interacts with a traffic harmonizer **144**, a polling server **145** and/or a policy management server **143**. Each of the various components can communicate with the client side proxy **175**, or other third party (e.g., application server/service provider **110** and/or other proxy **199**) and/or a reporting and usage analytics system. Some example components of the server side proxy **125** is described as follows:

Relay Server **142**: The relay server **142** is the routing agent in the distributed proxy architecture. The relay server **142** manages connections and communications with components on the client-side proxy **175** installed on devices and provides an administrative interface for reports, provisioning, platform setup, and so on.

Extended Caching Optimization Manager **179**: Similar to what are mentioned with regard to FIG. **1**A, the optimization manager **179** can perform, in conjunction with the optimization engine **177** (FIG. **1**E), signal optimization techniques, and more specifically, extended caching techniques with configurable parameters as disclosed herein. In some embodiments, such as the one shown in FIG. **1**F, the optimization manager **179** is coupled to the relay server **142** to receive relevant connection and communication information for performing the extended caching. Among others, the optimization manager **179** is also coupled to server cache **135** in carrying out the extended caching.

Notification Server **141**: The notification server **141** is a module able to connect to an operator's SMSC gateways and deliver SMS notifications to the client-side proxy **175**. SMS notifications can be used when an IP link is not currently active, in order to avoid the client-side proxy **175** from activating a connection over the wireless data channel, thus avoiding additional signaling traffic. However, if the IP connection happens to be open for some other traffic, the notification server **141** can use it for sending the notifications to the client-side proxy **175**. A user database **146** can store operational data including endpoint (MSISDN), organization and Notification server **141** gateway for each resource (URIs or URLs).

Traffic Harmonizer **144**: The traffic harmonizer **144** can be responsible for communication between the client-side proxy **175** and the polling server **145**. The traffic harmonizer **144** connects to the polling server **145** directly or through the data storage **130**, and to the client over any open or proprietary protocol such as the 7TP, implemented for traffic optimization. The traffic harmonizer **144** can be also responsible for traffic pipelining on the server side: if there's cached content in the database for the same client, this can be sent over to the client in one message.

Polling Server **145**: The polling server **145** can poll third party application servers on behalf of applications that are being optimized). If a change occurs (i.e. new data available) for an application, the polling server **145** can report to the traffic harmonizer **144** which in turn sends a notification message to the client-side proxy **175** for it to clear the cache and allow application to poll application server directly.

Policy Management Server **143**: The policy management server (PMS) **143** allows administrators to configure and store policies for the client-side proxies **175** (device clients). It also allows administrators to notify the client-side proxies **175** about policy changes. Using the policy management server **143**, each operator can configure the policies to work in the most efficient way for the unique characteristics of each particular mobile operator's network. In one embodiment, PMS **143** may include a policy management server database **147** for storing policies and other information.

Reporting and Usage Analytics Component: The Reporting and Usage Analytics component or system collects information from the client side **175** and/or from the server side **125**, and provides the tools for producing reports and usage analytics that operators can use for analyzing application signaling and data consumption.

Most mobile applications regularly poll their application servers to check for new data. Often there is no new data or the content has not changed, so the exchange of data through the mobile network is unnecessary. As the number of mobile phones and their applications increase, the amount of this needless polling grows. Since applications are not coordinated and poll at different times and intervals, any given phone may frequently generate signal traffic. This causes multiple unnecessary radio activations, consuming power and shortening battery life.

In one embodiment, the signaling optimizer reduces network requests to a minimum by caching content in the client and letting its own server poll for changes in the network. When a mobile phone's client side proxy (e.g., local proxy) **175** detects a recurring pattern for a resource, such as an email application, its response content is stored locally in a client cache so similar requests from that application get their response from the local cache, rather than signaling the network.

In another embodiment, systems and methods of intelligent alarm tracker and resource manipulator can be used to reduce network requests by consolidating or changing the timing of requests such that use of resources including network, battery, CPU, memory and the like can be reduced.

In some embodiments, features of the signaling optimizer and the intelligent alarm tracker and resource manipulator may be used together to obtain reduce resource usage by mobile applications on a mobile device.

FIG. **1G** illustrates an example diagram of a signaling optimizer of the distributed proxy and cache system.

As an example, someone who typically gets only 10 emails a day may have phone's email application poll the network for new email every 15 minutes, or 96 times a day, with around 90% or more of the polls resulting in the same response: there are no new emails. The client side proxy (e.g., local proxy) **175** can recognize this request—response pattern, and intercepts the application's poll requests, returning the locally cached response of "no new emails". This way the device radio is not turned on by this particular application, and the poll doesn't use any network resources. The server (e.g., host server **100**, proxy server **125**), located in the network, can monitor the email application server on behalf of the user's email application. When new email is available, the server can notify the user's client side proxy **175** to not use the cached "no new emails" response for the next poll request. Instead of going to the local client cache, the email application polls its application server over the network and receives the new content.

The signaling optimizer can be configured and managed using different rule sets for different device types, user types, wireless networks, and applications. Optimization rules can be updated at any time, so the changes can be applied immediately when an application upgrades or changes happen in the mobile network. The protocols that can be optimized include, but are not limited to: HTTP, HTTPS and DNS.

FIG. **1H** illustrates an example diagram of an example client-server architecture of the distributed proxy and cache system.

In the client-server architecture, the client side proxy **175** (e.g., local proxy) is residing on the mobile or client devices. The client side proxy **175** can communicate both directly to the Internet (usually via an operator proxy) and to the server side proxy (e.g., proxy server) **125**, or the host server **100**. The proxy server **125** communicates to the Internet and to the operator's SMSC **114**.

As depicted, the client side proxy **175** can send a request directly to the Internet. This can happen after requests have been analyzed to detect optimizable patterns, for example. The client side proxy **175** can, in one implementation, send a request to the server (e.g., host server **100**, proxy server **125**), for example, to initiate server polling, to reports logs or to get new configuration. The proxy server **125** can send a request to the Internet to, for example, validate cached content. In one implementation, the proxy server **125** can send a request to the SMSC **114**, for example, to send a cache invalidate message or policy update message to the client-side proxy **175**.

In one implementation, the client side proxy **175** may not maintain an open connection with the proxy server **125**, but may connect to the proxy server **125** only in case there's a need to start polling an origin server **110**, to report logs or to get new configuration. For signaling optimizer feature, the proxy server **125** can notify the client side proxy **175** when the content, that has been polled, has changed. The proxy server **125** can send a request to invalidate cache in the client side proxy **125**. When the application connects to that particular origin server (e.g., content server **110**) the next time, it can first fetch the latest content from the proxy server **125** and then directly connect to the origin server **110**. For the policy enforcer and/or the network protector features, the proxy server **125** can notify the client side proxy **175** when there's new configuration to be fetched from the server. When the proxy server **125** needs to communicate with the client side proxy **175**, it can use a connection that is already open for some other request. If the connection is not open, the proxy server **125** can send a notification (e.g., SMS) to the client side proxy **175**.

FIG. **1I** depicts an example diagram illustrating data flows between example client side components in a distributed proxy and cache system. Traffic from applications (e.g., App1, App2, App3 to AppN), client side proxy (e.g., local proxy) **175**, IP Routing Tables (e.g., in the Android Operating System Layer), Network Access Layer and Wireless Network are depicted.

In one implementation, non-optimized application traffic flow, such as traffic from App1, can completely bypass the client side proxy **175** components and proceed directly through the operating system layer (e.g., the Android OS layer) and Network Access Layer to the wireless network. Traffic that that is not optimized can include, but is not limited to: rich media, like video and audio, as well as traffic from networks and applications that has been configured to bypass optimization and traffic pending optimization, and the like. In one embodiment, all traffic can be configured to bypass the client side/server side proxy.

In another implementation, optimized application traffic, such as traffic from App2, can be redirected from the application to the client side proxy **175**. By default, this can be traffic on ports 80 (HTTP) and 53 (DNS), and selected traffic on port 443 (HTTPS), for example. However, traffic to other ports can be configured to be directed to the client side proxy.

In yet another implementation, traffic flow can be between the client side proxy **175** and the origin servers (e.g., content server **110**) via the Internet and/or between the client side proxy **175** and the server side proxy (e.g., proxy server) **125**.

FIG. **2**A depicts a block diagram illustrating an example of client-side components in a distributed proxy and cache system residing on a mobile device (e.g., wireless device) **250** that manages traffic in a wireless network (or broadband network) for resource conservation, content caching, traffic management, and/or signal optimization including extended caching optimization. The client-side proxy (or local proxy **275**) can further categorize mobile traffic and/or implement delivery policies based on application behavior, content priority, user activity, and/or user expectations.

The device **250**, which can be a portable or mobile device (e.g., any wireless device), such as a portable phone, generally includes, for example, a network interface **208** an operating system **204**, a context API **206**, and mobile applications which may be proxy-unaware **210** or proxy-aware **220**. Note that the device **250** is specifically illustrated in the example of FIG. **2** as a mobile device, such is not a limitation and that device **250** may be any wireless, broadband, portable/mobile or non-portable device able to receive, transmit signals to satisfy data requests over a network including wired or wireless networks (e.g., WiFi, cellular, Bluetooth, LAN, WAN, etc.).

The network interface **208** can be a networking module that enables the device **250** to mediate data in a network with an entity that is external to the host server **250**, through any known and/or convenient communications protocol supported by the host and the external entity. The network interface **208** can include one or more of a network adaptor card, a wireless network interface card (e.g., SMS interface, WiFi interface, interfaces for various generations of mobile communication standards including but not limited to 2G, 3G, 3.5G, 4G, LTE, etc.,), Bluetooth, or whether or not the connection is via a router, an access point, a wireless router, a switch, a multilayer switch, a protocol converter, a gateway, a bridge, a bridge router, a hub, a digital media receiver, and/or a repeater.

Device **250** can further include, client-side components of the distributed proxy and cache system which can include, a local proxy **275** (e.g., a mobile client of a mobile device) and a cache **285**. In one embodiment, the local proxy **275** includes a user activity module **215**, a proxy API **225**, a request/transaction manager **235**, a caching policy manager **245** having an application protocol module **248**, a traffic shaping engine **255**, and/or a connection manager **265**. The traffic shaping engine **255** may further include an alignment module **256** and/or a batching module **257**, the connection manager **265** may further include a radio controller **266**. The request/transaction manager **235** can further include an application behavior detector **236** and/or a prioritization engine **241**, the application behavior detector **236** may further include a pattern detector **237** and/or and application profile generator **239**. Additional or less components/modules/engines can be included in the local proxy **275** and each illustrated component.

As used herein, a "module," "a manager," a "handler," a "detector," an "interface," a "controller," a "normalizer," a "generator," an "invalidator," or an "engine" includes a general purpose, dedicated or shared processor and, typically, firmware or software modules that are executed by the processor. Depending upon implementation-specific or other considerations, the module, manager, handler, detector, interface, controller, normalizer, generator, invalidator, or engine can be centralized or its functionality distributed. The module, manager, handler, detector, interface, controller, normalizer, generator, invalidator, or engine can include general or special purpose hardware, firmware, or software embodied in a computer-readable (storage) medium for execution by the processor.

As used herein, a computer-readable medium or computer-readable storage medium is intended to include all mediums that are statutory (e.g., in the United States, under 35 U.S.C. §101), and to specifically exclude all mediums that are non-statutory in nature to the extent that the exclusion is necessary for a claim that includes the computer-readable (storage) medium to be valid. Known statutory computer-readable mediums include hardware (e.g., registers, random access memory (RAM), non-volatile (NV) storage, to name a few), but may or may not be limited to hardware.

In one embodiment, a portion of the distributed proxy and cache system for network traffic management resides in or is in communication with device **250**, including local proxy **275** (mobile client) and/or cache **285**. The local proxy **275** can provide an interface on the device **250** for users to access device applications and services including email, IM, voice mail, visual voicemail, feeds, Internet, games, productivity tools, or other applications, etc.

The proxy **275** is generally application independent and can be used by applications (e.g., both proxy-aware and proxy-unaware applications **210** and **220** and other mobile applications) to open TCP connections to a remote server (e.g., the server **100** in the examples of FIGS. **1B**-**1C** and/or server proxy **125**/**325** shown in the examples of FIG. **1B** and

FIG. **3**A). In some instances, the local proxy **275** includes a proxy API **225** which can be optionally used to interface with proxy-aware applications **220** (or applications (e.g., mobile applications) on a mobile device (e.g., any wireless device)).

The applications **210** and **220** can generally include any user application, widgets, software, HTTP-based application, web browsers, video or other multimedia streaming or downloading application, video games, social network applications, email clients, RSS management applications, application stores, document management applications, productivity enhancement applications, etc. The applications can be provided with the device OS, by the device manufacturer, by the network service provider, downloaded by the user, or provided by others.

One embodiment of the local proxy **275** includes or is coupled to a context API **206**, as shown. The context API **206** may be a part of the operating system **204** or device platform or independent of the operating system **204**, as illustrated. The operating system **204** can include any operating system including but not limited to, any previous, current, and/or future versions/releases of, Windows Mobile, iOS, Android, Symbian, Palm OS, Brew MP, Java 2 Micro Edition (J2ME), Blackberry, etc.

The context API **206** may be a plug-in to the operating system **204** or a particular client/application on the device **250**. The context API **206** can detect signals indicative of user or device activity, for example, sensing motion, gesture, device location, changes in device location, device backlight, keystrokes, clicks, activated touch screen, mouse click or detection of other pointer devices. The context API **206** can be coupled to input devices or sensors on the device **250** to identify these signals. Such signals can generally include input received in response to explicit user input at an input device/mechanism at the device **250** and/or collected from ambient signals/contextual cues detected at or in the vicinity of the device **250** (e.g., light, motion, piezoelectric, etc.).

In one embodiment, the user activity module **215** interacts with the context API **206** to identify, determine, infer, detect, compute, predict, and/or anticipate, characteristics of user activity on the device **250**. Various inputs collected by the context API **206** can be aggregated by the user activity module **215** to generate a profile for characteristics of user activity. Such a profile can be generated by the user activity module **215** with various temporal characteristics. For instance, user activity profile can be generated in real-time for a given instant to provide a view of what the user is doing or not doing at a given time (e.g., defined by a time window, in the last minute, in the last 30 seconds, etc.), a user activity profile can also be generated for a 'session' defined by an application or web page that describes the characteristics of user behavior with respect to a specific task they are engaged in on the device **250**, or for a specific time period (e.g., for the last 2 hours, for the last 5 hours).

Additionally, characteristic profiles can be generated by the user activity module **215** to depict a historical trend for user activity and behavior (e.g., 1 week, 1 mo., 2 mo., etc.). Such historical profiles can also be used to deduce trends of user behavior, for example, access frequency at different times of day, trends for certain days of the week (weekends or week days), user activity trends based on location data (e.g., IP address, GPS, or cell tower coordinate data) or changes in location data (e.g., user activity based on user location, or user activity based on whether the user is on the go, or traveling outside a home region, etc.) to obtain user activity characteristics.

In one embodiment, user activity module **215** can detect and track user activity with respect to applications, documents, files, windows, icons, and folders on the device **250**. For example, the user activity module **215** can detect when an application or window (e.g., a web browser or any other type of application) has been exited, closed, minimized, maximized, opened, moved into the foreground, or into the background, multimedia content playback, etc.

In one embodiment, characteristics of the user activity on the device **250** can be used to locally adjust behavior of the device (e.g., mobile device or any wireless device) to optimize its resource consumption such as battery/power consumption and more generally, consumption of other device resources including memory, storage, and processing power. In one embodiment, the use of a radio on a device can be adjusted based on characteristics of user behavior (e.g., by the radio controller **266** of the connection manager **265**) coupled to the user activity module **215**. For example, the radio controller **266** can turn the radio on or off, based on characteristics of the user activity on the device **250**. In addition, the radio controller **266** can adjust the power mode of the radio (e.g., to be in a higher power mode or lower power mode) depending on characteristics of user activity.

In one embodiment, characteristics of the user activity on device **250** can also be used to cause another device (e.g., other computers, a mobile device, a wireless device, or a non-portable device) or server (e.g., host server **100** and **300** in the examples of FIGS. **1**B-C and FIG. **3**A) which can communicate (e.g., via a cellular or other network) with the device **250** to modify its communication frequency with the device **250**. The local proxy **275** can use the characteristics information of user behavior determined by the user activity module **215** to instruct the remote device as to how to modulate its communication frequency (e.g., decreasing communication frequency, such as data push frequency if the user is idle, requesting that the remote device notify the device **250** if new data, changed, data, or data of a certain level of importance becomes available, etc.).

In one embodiment, the user activity module **215** can, in response to determining that user activity characteristics indicate that a user is active after a period of inactivity, request that a remote device (e.g., server host server **100** and **300** in the examples of FIGS. **1**B-C and FIG. **3**A) send the data that was buffered as a result of the previously decreased communication frequency.

In addition, or in alternative, the local proxy **275** can communicate the characteristics of user activity at the device **250** to the remote device (e.g., host server **100** and **300** in the examples of FIGS. **1**B-C and FIG. **3**A) and the remote device determines how to alter its own communication frequency with the device **250** for network resource conservation and conservation of device **250** resources.

One embodiment of the local proxy **275** further includes a request/transaction manager **235**, which can detect, identify, intercept, process, manage, data requests initiated on the device **250**, for example, by applications **210** and/or **220**, and/or directly/indirectly by a user request. The request/transaction manager **235** can determine how and when to process a given request or transaction, or a set of requests/transactions, based on transaction characteristics.

The request/transaction manager **235** can prioritize requests or transactions made by applications and/or users at the device **250**, for example by the prioritization engine **241**. Importance or priority of requests/transactions can be determined by the request/transaction manager **235** by applying a rule set, for example, according to time sensitivity of the transaction, time sensitivity of the content in the transaction, time criticality of the transaction, time criticality of the data transmitted in the transaction, and/or time criticality or importance of an application making the request.

In addition, transaction characteristics can also depend on whether the transaction was a result of user-interaction or other user-initiated action on the device (e.g., user interaction with a application (e.g., a mobile application)). In general, a time critical transaction can include a transaction resulting from a user-initiated data transfer, and can be prioritized as such. Transaction characteristics can also depend on the amount of data that will be transferred or is anticipated to be transferred as a result of the requested transaction. For example, the connection manager **265**, can adjust the radio mode (e.g., high power or low power mode via the radio controller **266**) based on the amount of data that will need to be transferred.

In addition, the radio controller **266**/connection manager **265** can adjust the radio power mode (high or low) based on time criticality/sensitivity of the transaction. The radio controller **266** can trigger the use of high power radio mode when a time-critical transaction (e.g., a transaction resulting from a user-initiated data transfer, an application running in the foreground, any other event meeting a certain criteria) is initiated or detected.

In general, the priorities can be set by default, for example, based on device platform, device manufacturer, operating system, etc. Priorities can alternatively or in additionally be set by the particular application; for example, the Facebook application (e.g., a mobile application) can set its own priorities for various transactions (e.g., a status update can be of higher priority than an add friend request or a poke request, a message send request can be of higher priority than a message delete request, for example), an email client or IM chat client may have its own configurations for priority. The prioritization engine **241** may include set of rules for assigning priority.

The prioritization engine **241** can also track network provider limitations or specifications on application or transaction priority in determining an overall priority status for a request/transaction. Furthermore, priority can in part or in whole be determined by user preferences, either explicit or implicit. A user, can in general, set priorities at different tiers, such as, specific priorities for sessions, or types, or applications (e.g., a browsing session, a gaming session, versus an IM chat session, the user may set a gaming session to always have higher priority than an IM chat session, which may have higher priority than web-browsing session). A user can set application-specific priorities, (e.g., a user may set Facebook-related transactions to have a higher priority than LinkedIn-related transactions), for specific transaction types (e.g., for all send message requests across all applications to have higher priority than message delete requests, for all calendar-related events to have a high priority, etc.), and/or for specific folders.

The prioritization engine **241** can track and resolve conflicts in priorities set by different entities. For example, manual settings specified by the user may take precedence over device OS settings, network provider parameters/limitations (e.g., set in default for a network service area, geographic locale, set for a specific time of day, or set based on service/fee type) may limit any user-specified settings and/or application-set priorities. In some instances, a manual synchronization request received from a user can override some, most, or all priority settings in that the requested synchronization is performed when requested, regardless of the individually assigned priority or an overall priority ranking for the requested action.

Priority can be specified and tracked internally in any known and/or convenient manner, including but not limited to, a binary representation, a multi-valued representation, a graded representation and all are considered to be within the scope of the disclosed technology.

TABLE 1

| Change (initiated on device) | Priority | Change (initiated on server) | Priority |
|---|---|---|---|
| Send email | High | Receive email | High |
| Delete email | Low | Edit email | Often not possible to sync (Low if possible) |
| (Un)read email | Low | | |
| Move message | Low | New email in deleted items | Low |
| Read more | High | | |
| Download attachment | High | Delete an email | Low |
| | | (Un)Read an email | Low |
| New Calendar event | High | Move messages | Low |
| Edit/change Calendar event | High | Any calendar change | High |
| | | Any contact change | High |
| Add a contact | High | Wipe/lock device | High |
| Edit a contact | High | Settings change | High |
| Search contacts | High | Any folder change | High |
| Change a setting | High | Connector restart | High (if no changes nothing is sent) |
| Manual send/receive | High | | |
| IM status change | Medium | Social Network Status Updates | Medium |
| Auction outbid or change notification | High | Severe Weather Alerts | High |
| Weather Updates | Low | News Updates | Low |

Table I above shows, for illustration purposes, some examples of transactions with examples of assigned priorities in a binary representation scheme. Additional assignments are possible for additional types of events, requests, transactions, and as previously described, priority assignments can be made at more or less granular levels, e.g., at the session level or at the application level, etc.

As shown by way of example in the above table, in general, lower priority requests/transactions can include, updating message status as being read, unread, deleting of messages, deletion of contacts; higher priority requests/transactions, can in some instances include, status updates, new IM chat message, new email, calendar event update/cancellation/deletion, an event in a mobile gaming session, or other entertainment related events, a purchase confirmation through a web purchase or online, request to load additional or download content, contact book related events, a transaction to change a device setting, location-aware or location-based events/transactions, or any other events/request/transactions initiated by a user or where the user is known to be, expected to be, or suspected to be waiting for a response, etc.

Inbox pruning events (e.g., email, or any other types of messages), are generally considered low priority and absent other impending events, generally will not trigger use of the radio on the device **250**. Specifically, pruning events to remove old email or other content can be 'piggy backed' with other communications if the radio is not otherwise on, at the time of a scheduled pruning event. For example, if the user has preferences set to 'keep messages for 7 days old,' then instead of powering on the device radio to initiate a message delete from the device **250** the moment that the message has exceeded 7 days old, the message is deleted when the radio is powered on next. If the radio is already on, then pruning may occur as regularly scheduled.

The request/transaction manager **235**, can use the priorities for requests (e.g., by the prioritization engine **241**) to manage outgoing traffic from the device **250** for resource optimization (e.g., to utilize the device radio more efficiently for battery conservation). For example, transactions/requests below a certain priority ranking may not trigger use of the radio on the device **250** if the radio is not already switched on, as controlled by the connection manager **265**. In contrast, the radio controller **266** can turn on the radio such a request can be sent when a request for a transaction is detected to be over a certain priority level.

In one embodiment, priority assignments (such as that determined by the local proxy **275** or another device/entity) can be used cause a remote device to modify its communication with the frequency with the mobile device or wireless device. For example, the remote device can be configured to send notifications to the device **250** when data of higher importance is available to be sent to the mobile device or wireless device.

In one embodiment, transaction priority can be used in conjunction with characteristics of user activity in shaping or managing traffic, for example, by the traffic shaping engine **255**. For example, the traffic shaping engine **255** can, in response to detecting that a user is dormant or inactive, wait to send low priority transactions from the device **250**, for a period of time. In addition, the traffic shaping engine **255** can allow multiple low priority transactions to accumulate for batch transferring from the device **250** (e.g., via the batching module **257**). In one embodiment, the priorities can be set, configured, or readjusted by a user. For example, content depicted in Table I in the same or similar form can be accessible in a user interface on the device **250** and for example, used by the user to adjust or view the priorities.

The batching module **257** can initiate batch transfer based on certain criteria. For example, batch transfer (e.g., of multiple occurrences of events, some of which occurred at different instances in time) may occur after a certain number of low priority events have been detected, or after an amount of time elapsed after the first of the low priority event was initiated. In addition, the batching module **257** can initiate batch transfer of the cumulated low priority events when a higher priority event is initiated or detected at the device **250**. Batch transfer can otherwise be initiated when radio use is triggered for another reason (e.g., to receive data from a remote device such as host server **100** or **300**). In one embodiment, an impending pruning event (pruning of an inbox), or any other low priority events, can be executed when a batch transfer occurs.

In general, the batching capability can be disabled or enabled at the event/transaction level, application level, or session level, based on any one or combination of the following: user configuration, device limitations/settings, manufacturer specification, network provider parameters/limitations, platform-specific limitations/settings, device OS settings, etc. In one embodiment, batch transfer can be initiated when an application/window/file is closed out, exited, or moved into the background; users can optionally be prompted before initiating a batch transfer; users can also manually trigger batch transfers.

In one embodiment, the local proxy **275** locally adjusts radio use on the device **250** by caching data in the cache **285**. When requests or transactions from the device **250** can be satisfied by content stored in the cache **285**, the radio controller **266** need not activate the radio to send the request to a remote entity (e.g., the host server **100**, **300**, as shown in FIG. **1**B and FIG. **3**A or a content provider/application server such as the server/provider **110** shown in the examples of FIG. **1**B and FIG. **1**C). As such, the local proxy **275** can use the local cache **285** and the cache policy manager **245** to locally store data for satisfying data requests to eliminate or reduce the use of the device radio for conservation of network resources and device battery consumption.

In leveraging the local cache, once the request/transaction manager **225** intercepts a data request by an application on the device **250**, the local repository **285** can be queried to determine if there is any locally stored response, and also determine whether the response is valid. When a valid response is available in the local cache **285**, the response can be provided to the application on the device **250** without the device **250** needing to access the cellular network or wireless broadband network.

If a valid response is not available, the local proxy **275** can query a remote proxy (e.g., the server proxy **325** of FIG. **3**A) to determine whether a remotely stored response is valid. If so, the remotely stored response (e.g., which may be stored on the server cache **135** or optional caching server **199** shown in the example of FIG. **1**C) can be provided to the mobile device, possibly without the mobile device **250** needing to access the cellular network, thus relieving consumption of network resources.

If a valid cache response is not available, or if cache responses are unavailable for the intercepted data request, the local proxy **275**, for example, the caching policy manager **245**, can send the data request to a remote proxy (e.g., server proxy **325** of FIG. **3**A) which forwards the data request to a content source (e.g., application server/content provider **110** of FIG. **1**B) and a response from the content source can be provided through the remote proxy, as will be further described in the description associated with the example host server **300** of FIG. **3**A. The cache policy manager **245** can manage or process requests that use a variety of protocols, including but not limited to HTTP, HTTPS, IMAP, POP, SMTP, XMPP, and/or ActiveSync. The caching policy manager **245** can locally store responses for data requests in the local database **285** as cache entries, for subsequent use in satisfying same or similar data requests.

The caching policy manager **245** can request that the remote proxy monitor responses for the data request and the remote proxy can notify the device **250** when an unexpected response to the data request is detected. In such an event, the cache policy manager **245** can erase or replace the locally stored response(s) on the device **250** when notified of the unexpected response (e.g., new data, changed data, additional data, etc.) to the data request. In one embodiment, the caching policy manager **245** is able to detect or identify the protocol used for a specific request, including but not limited to HTTP, HTTPS, IMAP, POP, SMTP, XMPP, and/or ActiveSync. In one embodiment, application specific handlers (e.g., via the application protocol module **246** of the caching policy manager **245**) on the local proxy **275** allows for optimization of any protocol that can be port mapped to a handler in the distributed proxy (e.g., port mapped on the proxy server **325** in the example of FIG. **3**A).

In one embodiment, the local proxy **275** notifies the remote proxy such that the remote proxy can monitor responses received for the data request from the content source for changed results prior to returning the result to the device **250**, for example, when the data request to the content source has yielded same results to be returned to the mobile device. In general, the local proxy **275** can simulate application server responses for applications on the device **250**, using locally cached content. This can prevent utilization of the cellular network for transactions where new/

changed data is not available, thus freeing up network resources and preventing network congestion.

In one embodiment, the local proxy **275** includes an application behavior detector **236** to track, detect, observe, monitor, applications (e.g., proxy-aware and/or unaware applications **210** and **220**) accessed or installed on the device **250**. Application behaviors, or patterns in detected behaviors (e.g., via the pattern detector **237**) of one or more applications accessed on the device **250** can be used by the local proxy **275** to optimize traffic in a wireless network needed to satisfy the data needs of these applications.

For example, based on detected behavior of multiple applications, the traffic shaping engine **255** can align content requests made by at least some of the applications over the network (wireless network) (e.g., via the alignment module **256**). The alignment module **256** can delay or expedite some earlier received requests to achieve alignment. When requests are aligned, the traffic shaping engine **255** can utilize the connection manager to poll over the network to satisfy application data requests. Content requests for multiple applications can be aligned based on behavior patterns or rules/settings including, for example, content types requested by the multiple applications (audio, video, text, etc.), device (e.g., mobile or wireless device) parameters, and/or network parameters/traffic conditions, network service provider constraints/specifications, etc.

In one embodiment, the pattern detector **237** can detect recurrences in application requests made by the multiple applications, for example, by tracking patterns in application behavior. A tracked pattern can include, detecting that certain applications, as a background process, poll an application server regularly, at certain times of day, on certain days of the week, periodically in a predictable fashion, with a certain frequency, with a certain frequency in response to a certain type of event, in response to a certain type user query, frequency that requested content is the same, frequency with which a same request is made, interval between requests, applications making a request, or any combination of the above, for example.

Such recurrences can be used by traffic shaping engine **255** to offload polling of content from a content source (e.g., from an application server/content provider **110** of FIG. **1**A) that would result from the application requests that would be performed at the mobile device or wireless device **250** to be performed instead, by a proxy server (e.g., proxy server **125** of FIG. **1**C or proxy server **325** of FIG. **3**A) remote from the device **250**. Traffic shaping engine **255** can decide to offload the polling when the recurrences match a rule. For example, there are multiple occurrences or requests for the same resource that have exactly the same content, or returned value, or based on detection of repeatable time periods between requests and responses such as a resource that is requested at specific times during the day. The offloading of the polling can decrease the amount of bandwidth consumption needed by the mobile device **250** to establish a wireless (cellular or other wireless broadband) connection with the content source for repetitive content polls.

As a result of the offloading of the polling, locally cached content stored in the local cache **285** can be provided to satisfy data requests at the device **250**, when content change is not detected in the polling of the content sources. As such, when data has not changed, application data needs can be satisfied without needing to enable radio use or occupying cellular bandwidth in a wireless network. When data has changed and/or new data has been received, the remote

entity to which polling is offloaded, can notify the device **250**. The remote entity may be the host server **300** as shown in the example of FIG. **3**A.

In one embodiment, the local proxy **275** can mitigate the need/use of periodic keep-alive messages (heartbeat messages) to maintain TCP/IP connections, which can consume significant amounts of power thus having detrimental impacts on mobile device battery life. The connection manager **265** in the local proxy (e.g., the heartbeat manager **267**) can detect, identify, and intercept any or all heartbeat (keep-alive) messages being sent from applications.

The heartbeat manager **267** can prevent any or all of these heartbeat messages from being sent over the cellular, or other network, and instead rely on the server component of the distributed proxy system (e.g., shown in FIG. **1**C) to generate and send the heartbeat messages to maintain a connection with the backend (e.g., application server/provider **110** in the example of FIG. **1**A).

The local proxy **275** generally represents any one or a portion of the functions described for the individual managers, modules, and/or engines. The local proxy **275** and device **250** can include additional or less components; more or less functions can be included, in whole or in part, without deviating from the novel art of the disclosure.

FIG. **2**B depicts a block diagram illustrating a further example of components in the cache system shown in the example of FIG. **2**A which is capable of caching and adapting caching strategies for mobile application behavior and/or network conditions.

In one embodiment, the caching policy manager **245** includes a metadata generator **203**, a cache look-up engine **205**, a cache appropriateness decision engine **246**, a poll schedule generator **247**, an application protocol module **248**, a cache or connect selection engine **249** and/or a local cache invalidator **244**. The cache appropriateness decision engine **246** can further include a timing predictor **246***a*, a content predictor **246***b*, a request analyzer **246***c*, and/or a response analyzer **246***d*, and the cache or connect selection engine **249** includes a response scheduler **249***a*. The metadata generator **203** and/or the cache look-up engine **205** are coupled to the cache **285** (or local cache) for modification or addition to cache entries or querying thereof.

The cache look-up engine **205** may further include an ID or URI filter **205***a*, the local cache invalidator **244** may further include a TTL manager **244***a*, and the poll schedule generator **247** may further include a schedule update engine **247***a* and/or a time adjustment engine **247***b*. One embodiment of caching policy manager **245** includes an application cache policy repository **243**. In one embodiment, the application behavior detector **236** includes a pattern detector **237**, a poll interval detector **238**, an application profile generator **239**, and/or a priority engine **241**. The poll interval detector **238** may further include a long poll detector **238***a* having a response/request tracking engine **238***b*. The poll interval detector **238** may further include a long poll hunting detector **238***c*. The application profile generator **239** can further include a response delay interval tracker **239***a*.

The pattern detector **237**, application profile generator **239**, and the priority engine **241** were also described in association with the description of the pattern detector shown in the example of FIG. **2**A. One embodiment further includes an application profile repository **242** which can be used by the local proxy **275** to store information or metadata regarding application profiles (e.g., behavior, patterns, type of HTTP requests, etc.)

The cache appropriateness decision engine **246** can detect, assess, or determine whether content from a content source (e.g., application server/content provider **110** in the example of FIG. **1**B) with which a mobile device **250** interacts and has content that may be suitable for caching. For example, the decision engine **246** can use information about a request and/or a response received for the request initiated at the mobile device **250** to determine cacheability, potential cacheability, or non-cacheability. In some instances, the decision engine **246** can initially verify whether a request is directed to a blacklisted destination or whether the request itself originates from a blacklisted client or application. If so, additional processing and analysis may not be performed by the decision engine **246** and the request may be allowed to be sent over the air to the server to satisfy the request. The black listed destinations or applications/clients (e.g., mobile applications) can be maintained locally in the local proxy (e.g., in the application profile repository **242**) or remotely (e.g., in the proxy server **325** or another entity).

In one embodiment, the decision engine **246**, for example, via the request analyzer **246***c*, collects information about an application or client request generated at the mobile device **250**. The request information can include request characteristics information including, for example, request method. For example, the request method can indicate the type of HTTP request generated by the mobile application or client. In one embodiment, response to a request can be identified as cacheable or potentially cacheable if the request method is a GET request or POST request. Other types of requests (e.g., OPTIONS, HEAD, PUT, DELETE, TRACE, or CONNECT) may or may not be cached. In general, HTTP requests with uncacheable request methods will not be cached.

Request characteristics information can further include information regarding request size, for example. Responses to requests (e.g., HTTP requests) with body size exceeding a certain size will not be cached. For example, cacheability can be determined if the information about the request indicates that a request body size of the request does not exceed a certain size. In some instances, the maximum cacheable request body size can be set to 8092 bytes. In other instances, different values may be used, dependent on network capacity or network operator specific settings, for example.

In some instances, content from a given application server/content provider (e.g., the server/content provider **110** of FIG. **1**C) is determined to be suitable for caching based on a set of criteria, for example, criteria specifying time criticality of the content that is being requested from the content source. In one embodiment, the local proxy (e.g., the local proxy **175** or **275** of FIG. **1**C and FIG. **2**A) applies a selection criteria to store the content from the host server which is requested by an application as cached elements in a local cache on the mobile device to satisfy subsequent requests made by the application.

The cache appropriateness decision engine **246**, further based on detected patterns of requests sent from the mobile device **250** (e.g., by a mobile application or other types of clients on the device **250**) and/or patterns of received responses, can detect predictability in requests and/or responses. For example, the request characteristics information collected by the decision engine **246**, (e.g., the request analyzer **246***c*) can further include periodicity information between a request and other requests generated by a same client on the mobile device or other requests directed to the same host (e.g., with similar or same identifier parameters).

Periodicity can be detected, by the decision engine **246** or the request analyzer **246***c*, when the request and the other requests generated by the same client occur at a fixed rate or nearly fixed rate, or at a dynamic rate with some identifiable or partially or wholly reproducible changing pattern. If the requests are made with some identifiable pattern (e.g., regular intervals, intervals having a detectable pattern, or trend (e.g., increasing, decreasing, constant, etc.) the timing predictor **246***a* can determine that the requests made by a given application on a device is predictable and identify it to be potentially appropriate for caching, at least from a timing standpoint.

An identifiable pattern or trend can generally include any application or client behavior which may be simulated either locally, for example, on the local proxy **275** on the mobile device **250** or simulated remotely, for example, by the proxy server **325** on the host **300**, or a combination of local and remote simulation to emulate application behavior.

In one embodiment, the decision engine **246**, for example, via the response analyzer **246***d*, can collect information about a response to an application or client request generated at the mobile device **250**. The response is typically received from a server or the host of the application (e.g., mobile application) or client which sent the request at the mobile device **250**. In some instances, the mobile client or application can be the mobile version of an application (e.g., social networking, search, travel management, voicemail, contact manager, email) or a web site accessed via a web browser or via a desktop client.

For example, response characteristics information can include an indication of whether transfer encoding or chunked transfer encoding is used in sending the response. In some instances, responses to HTTP requests with transfer encoding or chunked transfer encoding are not cached, and therefore are also removed from further analysis. The rationale here is that chunked responses are usually large and non-optimal for caching, since the processing of these transactions may likely slow down the overall performance. Therefore, in one embodiment, cacheability or potential for cacheability can be determined when transfer encoding is not used in sending the response.

In addition, the response characteristics information can include an associated status code of the response which can be identified by the response analyzer **246***d*. In some instances, HTTP responses with uncacheable status codes are typically not cached. The response analyzer **246***d* can extract the status code from the response and determine whether it matches a status code which is cacheable or uncacheable. Some cacheable status codes include by way of example: 200—OK, 301—Redirect, 302—Found, 303—See other, 304—Not Modified, 307 Temporary Redirect, or 500—Internal server error. Some uncacheable status codes can include, for example, 403—Forbidden or 404—Not found.

In one embodiment, cacheability or potential for cacheability can be determined if the information about the response does not indicate an uncacheable status code or indicates a cacheable status code. If the response analyzer **246***d* detects an uncacheable status code associated with a given response, the specific transaction (request/response pair) may be eliminated from further processing and determined to be uncacheable on a temporary basis, a semi-permanent, or a permanent basis. If the status code indicates cacheability, the transaction (e.g., request and/or response pair) may be subject to further processing and analysis to confirm cacheability.

Response characteristics information can also include response size information. In general, responses can be cached locally at the mobile device **250** if the responses do not exceed a certain size. In some instances, the default maximum cached response size is set to 115 KB. In other instances, the max cacheable response size may be different and/or dynamically adjusted based on operating conditions, network conditions, network capacity, user preferences, network operator requirements, or other application-specific, user specific, and/or device-specific reasons. In one embodiment, the response analyzer **246*d*** can identify the size of the response, and cacheability or potential for cacheability can be determined if a given threshold or max value is not exceeded by the response size.

Furthermore, response characteristics information can include response body information for the response to the request and other response to other requests generated by a same client on the mobile device, or directed to a same content host or application server. The response body information for the response and the other responses can be compared, for example, by the response analyzer **246*d***, to prevent the caching of dynamic content (or responses with content that changes frequently and cannot be efficiently served with cache entries, such as financial data, stock quotes, news feeds, real-time sporting event activities, etc.), such as content that would no longer be relevant or up-to-date if served from cached entries.

The cache appropriateness decision engine **246** (e.g., the content predictor **246*b***) can definitively identify repeatability or identify indications of repeatability, potential repeatability, or predictability in responses received from a content source (e.g., the content host/application server **110** shown in the example of FIG. **1**C). Repeatability can be detected by, for example, tracking at least two responses received from the content source and determines if the two responses are the same. For example, cacheability can be determined, by the response analyzer **246*d***, if the response body information for the response and the other responses sent by the same mobile client or directed to the same host/server are same or substantially the same. The two responses may or may not be responses sent in response to consecutive requests. In one embodiment, hash values of the responses received for requests from a given application are used to determine repeatability of content (with or without heuristics) for the application in general and/or for the specific request. Additional same responses may be required for some applications or under certain circumstances.

Repeatability in received content need not be 100% ascertained. For example, responses can be determined to be repeatable if a certain number or a certain percentage of responses are the same, or similar. The certain number or certain percentage of same/similar responses can be tracked over a select period of time, set by default or set based on the application generating the requests (e.g., whether the application is highly dynamic with constant updates or less dynamic with infrequent updates). Any indicated predictability or repeatability, or possible repeatability, can be utilized by the distributed system in caching content to be provided to a requesting application or client on the mobile device **250**.

In one embodiment, for a long poll type request, the local proxy **175** can begin to cache responses on a third request when the response delay times for the first two responses are the same, substantially the same, or detected to be increasing in intervals. In general, the received responses for the first two responses should be the same, and upon verifying that the third response received for the third request is the same (e.g., if R0=R1=R2), the third response can be locally cached on the mobile device. Less or more same responses may be required to begin caching, depending on the type of application, type of data, type of content, user preferences, or carrier/network operator specifications.

Increasing response delays with same responses for long polls can indicate a hunting period (e.g., a period in which the application/client on the mobile device is seeking the longest time between a request and response that a given network will allow), as detected by the long poll hunting detector **238*c*** of the application behavior detector **236**.

An example can be described below using T0, T1, T2, where T indicates the delay time between when a request is sent and when a response (e.g., the response header) is detected/received for consecutive requests:

T0=Response0(t)−Request0(t)=180 s. (+/− tolerance)

T1=Response1(t)−Request1(t)=240 s. (+/− tolerance)

T2=Response2(t)−Request2(t)=500 s. (+/− tolerance)

In the example timing sequence shown above, T0<T1<T2, this may indicate a hunting pattern for a long poll when network timeout has not yet been reached or exceeded. Furthermore, if the responses R0, R1, and R2 received for the three requests are the same, R2 can be cached. In this example, R2 is cached during the long poll hunting period without waiting for the long poll to settle, thus expediting response caching (e.g., this is optional accelerated caching behavior which can be implemented for all or select applications).

As such, the local proxy **275** can specify information that can be extracted from the timing sequence shown above (e.g., polling schedule, polling interval, polling type) to the proxy server and begin caching and to request the proxy server to begin polling and monitoring the source (e.g., using any of T0, T1, T2 as polling intervals but typically T2, or the largest detected interval without timing out, and for which responses from the source is received will be sent to the proxy server **325** of FIG. **3**A for use in polling the content source (e.g., application server/service provider **310**)).

However, if the time intervals are detected to be getting shorter, the application (e.g., mobile application)/client may still be hunting for a time interval for which a response can be reliably received from the content source (e.g., application/server server/provider **110** or **310**), and as such caching typically should not begin until the request/response intervals indicate the same time interval or an increasing time interval, for example, for a long poll type request.

An example of handling a detected decreasing delay can be described below using T0, T1, T2, T3, and T4 where T indicates the delay time between when a request is sent and when a response (e.g., the response header) is detected/received for consecutive requests:

T0=Response0(t)−Request0(t)=160 s. (+/− tolerance)

T1=Response1(t)−Request1(t)=240 s. (+/− tolerance)

T2=Response2(t)−Request2(t)=500 s. (+/− tolerance)

T3=Time out at 700 s. (+/− tolerance)

T4=Response4(t)−Request4(t)=600 (+/− tolerance)

If a pattern for response delays T1<T2<T3>T4 is detected, as shown in the above timing sequence (e.g., detected by the long poll hunting detector **238*c*** of the application behavior detector **236**), it can be determined that T3 likely exceeded the network time out during a long poll hunting period. In Request 3, a response likely was not received since the connection was terminated by the network, application, server, or other reason before a response was sent or available. On Request 4 (after T4), if a response (e.g., Response 4) is detected or received, the local proxy **275** can then use the response for caching (if the content repeatability condition is met). The local proxy can also use T4 as the poll interval in the polling schedule set for the proxy server to monitor/poll the content source.

Note that the above description shows that caching can begin while long polls are in hunting mode in the event of detecting increasing response delays, as long as responses are received and not timed out for a given request. This can be referred to as the optional accelerated caching during long poll hunting. Caching can also begin after the hunting mode (e.g., after the poll requests have settled to a constant or near constant delay value) has completed. Note that hunting may or may not occur for long polls and when hunting occurs; the proxy **275** can generally detect this and determine whether to begin to cache during the hunting period (increasing intervals with same responses) or wait until the hunt settles to a stable value.

In one embodiment, the timing predictor **246***a* of the cache appropriateness decision engine **246** can track timing of responses received from outgoing requests from an application (e.g., mobile application) or client to detect any identifiable patterns which can be partially wholly reproducible, such that locally cached responses can be provided to the requesting client on the mobile device **250** in a manner that simulates content source (e.g., application server/content provider **110** or **310**) behavior. For example, the manner in which (e.g., from a timing standpoint) responses or content would be delivered to the requesting application/client on the device **250**. This ensures preservation of user experience when responses to application or mobile client requests are served from a local and/or remote cache instead of being retrieved/received directly from the content source (e.g., application, content provider **110** or **310**).

In one embodiment, the decision engine **246** or the timing predictor **246***a* determines the timing characteristics a given application (e.g., mobile application) or client from, for example, the request/response tracking engine **238***b* and/or the application profile generator **239** (e.g., the response delay interval tracker **239***a*). Using the timing characteristics, the timing predictor **246***a* determines whether the content received in response to the requests are suitable or are potentially suitable for caching. For example, poll request intervals between two consecutive requests from a given application can be used to determine whether request intervals are repeatable (e.g., constant, near constant, increasing with a pattern, decreasing with a pattern, etc.) and can be predicted and thus reproduced at least some of the times either exactly or approximated within a tolerance level.

In some instances, the timing characteristics of a given request type for a specific application, for multiple requests of an application, or for multiple applications can be stored in the application profile repository **242**. The application profile repository **242** can generally store any type of information or metadata regarding application request/response characteristics including timing patterns, timing repeatability, content repeatability, etc.

The application profile repository **242** can also store metadata indicating the type of request used by a given application (e.g., long polls, long-held HTTP requests, HTTP streaming, push, COMET push, etc.) Application profiles indicating request type by applications can be used when subsequent same/similar requests are detected, or when requests are detected from an application which has already been categorized. In this manner, timing characteristics for the given request type or for requests of a specific application which has been tracked and/or analyzed, need not be reanalyzed.

Application profiles can be associated with a time-to-live (e.g., or a default expiration time). The use of an expiration time for application profiles, or for various aspects of an

application or request's profile can be used on a case by case basis. The time-to-live or actual expiration time of application profile entries can be set to a default value or determined individually, or a combination thereof. Application profiles can also be specific to wireless networks, physical networks, network operators, or specific carriers.

One embodiment includes an application blacklist manager **201**. The application blacklist manager **201** can be coupled to the application cache policy repository **243** and can be partially or wholly internal to local proxy or the caching policy manager **245**. Similarly, the blacklist manager **201** can be partially or wholly internal to local proxy or the application behavior detector **236**. The blacklist manager **201** can aggregate, track, update, manage, adjust, or dynamically monitor a list of destinations of servers/host that are 'blacklisted,' or identified as not cached, on a permanent or temporary basis. The blacklist of destinations, when identified in a request, can potentially be used to allow the request to be sent over the (cellular) network for servicing. Additional processing on the request may not be performed since it is detected to be directed to a blacklisted destination.

Blacklisted destinations can be identified in the application cache policy repository **243** by address identifiers including specific URIs or patterns of identifiers including URI patterns. In general, blacklisted destinations can be set by or modified for any reason by any party including the user (owner/user of mobile device **250**), operating system/mobile platform of device **250**, the destination itself, network operator (of cellular network), Internet service provider, other third parties, or according to a list of destinations for applications known to be uncacheable/not suited for caching. Some entries in the blacklisted destinations may include destinations aggregated based on the analysis or processing performed by the local proxy (e.g., cache appropriateness decision engine **246**).

For example, applications or mobile clients on the mobile device for which responses have been identified as non-suitable for caching can be added to the blacklist Their corresponding hosts/servers may be added in addition to or in lieu of an identification of the requesting application/client on the mobile device **250**. Some or all of such clients identified by the proxy system can be added to the blacklist. For example, for all application clients or applications that are temporarily identified as not being suitable for caching, only those with certain detected characteristics (based on timing, periodicity, frequency of response content change, content predictability, size, etc.) can be blacklisted.

The blacklisted entries may include a list of requesting applications or requesting clients on the mobile device (rather than destinations) such that, when a request is detected from a given application or given client, it may be sent through the network for a response, since responses for blacklisted clients/applications are in most circumstances not cached.

A given application profile may also be treated or processed differently (e.g., different behavior of the local proxy **275** and the remote proxy **325**) depending on the mobile account associated with a mobile device from which the application is being accessed. For example, a higher paying account, or a premier account may allow more frequent access of the wireless network or higher bandwidth allowance thus affecting the caching policies implemented between the local proxy **275** and proxy server **325** with an emphasis on better performance compared to conservation of resources. A given application profile may also be treated or processed differently under different wireless network conditions (e.g., based on congestion or network outage, etc.).

Note that cache appropriateness can be determined, tracked, and managed for multiple clients or applications on the mobile device **250**. Cache appropriateness can also be determined for different requests or request types initiated by a given client or application on the mobile device **250**. The caching policy manager **245**, along with the timing predictor **246***a* and/or the content predictor **246***b* which heuristically determines or estimates predictability or potential predictability, can track, manage and store cacheability information for various application or various requests for a given application. Cacheability information may also include conditions (e.g., an application can be cached at certain times of the day, or certain days of the week, or certain requests of a given application can be cached, or all requests with a given destination address can be cached) under which caching is appropriate which can be determined and/or tracked by the cache appropriateness decision engine **246** and stored and/or updated when appropriate in the application cache policy repository **243** coupled to the cache appropriateness decision engine **246**.

The information in the application cache policy repository **243** regarding cacheability of requests, applications, and/or associated conditions can be used later on when same requests are detected. In this manner, the decision engine **246** and/or the timing and content predictors **246***a/b* need not track and reanalyze request/response timing and content characteristics to make an assessment regarding cacheability. In addition, the cacheability information can in some instances be shared with local proxies of other mobile devices by way of direct communication or via the host server (e.g., proxy server **325** of host server **300**).

For example, cacheability information detected by the local proxy **275** on various mobile devices can be sent to a remote host server or a proxy server **325** on the host server (e.g., host server **300** or proxy server **325** shown in the example of FIG. **3**A, host **100** and proxy server **125** in the example of FIGS. **1**B-C). The remote host or proxy server can then distribute the information regarding application-specific, request-specific cacheability information and/or any associated conditions to various mobile devices or their local proxies in a wireless network or across multiple wireless networks (same service provider or multiple wireless service providers) for their use.

In general, the selection criteria for caching can further include, by way of example but not limitation, the state of the mobile device indicating whether the mobile device is active or inactive, network conditions, and/or radio coverage statistics. The cache appropriateness decision engine **246** can in any one or any combination of the criteria, and in any order, identifying sources for which caching may be suitable.

Once application servers/content providers having identified or detected content that is potentially suitable for local caching on the mobile device **250**, the cache policy manager **245** can proceed to cache the associated content received from the identified sources by storing content received from the content source as cache elements in a local cache (e.g., local cache **185** or **285** shown in the examples of FIG. **1**B**1**C and FIG. **2**A, respectively) on the mobile device **250**.

The response can be stored in the cache **285** (e.g., also referred as the local cache) as a cache entry. In addition to the response to a request, the cached entry can include response metadata having additional information regarding caching of the response. The metadata may be generated by

the metadata generator **203** and can include, for example, timing data such as the access time of the cache entry or creation time of the cache entry. Metadata can include additional information, such as any information suited for use in determining whether the response stored as the cached entry is used to satisfy the subsequent response. For example, metadata information can further include, request timing history (e.g., including request time, request start time, request end time), hash of the request and/or response, time intervals or changes in time intervals, etc.

The cache entry is typically stored in the cache **285** in association with a time-to-live (TTL), which for example may be assigned or determined by the TTL manager **244***a* of the cache invalidator **244**. The time-to-live of a cache entry is the amount of time the entry is persisted in the cache **285** regardless of whether the response is still valid or relevant for a given request or client/application on the mobile device **250**. For example, if the time-to-live of a given cache entry is set to 12 hours, the cache entry is purged, removed, or otherwise indicated as having exceeded the time-to-live, even if the response body contained in the cache entry is still current and applicable for the associated request.

A default time-to-live can be automatically used for all entries unless otherwise specified (e.g., by the TTL manager **244***a*), or each cache entry can be created with its individual TTL (e.g., determined by the TTL manager **244***a* based on various dynamic or static criteria). Note that each entry can have a single time-to-live associated with both the response data and any associated metadata. In some instances, the associated metadata may have a different time-to-live (e.g., a longer time-to-live) than the response data.

The content source having content for caching can, in addition or in alternate, be identified to a proxy server (e.g., proxy server **125** or **325** shown in the examples of FIGS. **1**B-**1**C and FIG. **3**A, respectively) remote from and in wireless communication with the mobile device **250** such that the proxy server can monitor the content source (e.g., application server/content provider **110**) for new or changed data. Similarly, the local proxy (e.g., the local proxy **175** or **275** of FIGS. **1**B-**1**C and FIG. **2**A, respectively) can identify to the proxy server that content received from a specific application server/content provider is being stored as cached elements in the local cache **285**.

Once content has been locally cached, the cache policy manager **245**, upon receiving future polling requests to contact the application server/content host (e.g., **110** or **310**), can retrieve the cached elements from the local cache to respond to the polling request made at the mobile device **250** such that a radio of the mobile device is not activated to service the polling request. For example, the cache look-up engine **205** can query the cache **285** to identify the response to be served to a response. The response can be served from the cache in response to identifying a matching cache entry and also using any metadata stored with the response in the cache entry. The cache entries can be queried by the cache look-up engine using a URI of the request or another type of identifier (e.g., via the ID or URI filter **205***a*). The cache-lookup engine **205** can further use the metadata (e.g., extract any timing information or other relevant information) stored with the matching cache entry to determine whether response is still suited for use in being served to a current request.

Note that the cache-look-up can be performed by the engine **205** using one or more of various multiple strategies. In one embodiment, multiple cook-up strategies can be executed sequentially on each entry store din the cache **285**, until at least one strategy identifies a matching cache entry.

The strategy employed to performing cache look-up can include a strict matching criteria or a matching criteria which allows for non-matching parameters.

For example, the look-up engine **205** can perform a strict matching strategy which searches for an exact match between an identifier (e.g., a URI for a host or resource) referenced in a present request for which the proxy is attempting to identify a cache entry and an identifier stored with the cache entries. In the case where identifiers include URIs or URLs, the matching algorithm for strict matching will search for a cache entry where all the parameters in the URLs match. For example:

Example 1.

Cache contains entry for <URL>/products/

Request is being made to <URL>/products/

Strict strategy will find a match, since both URIs are same.

Example 2.

Cache contains entry for <URL>/products/?query=all

Request is being made to <URL>/products/?query=sub

Under the strict strategy outlined above, a match will not be found since the URIs differ in the query parameter.

In another example strategy, the look-up engine **205** looks for a cache entry with an identifier that partially matches the identifier references in a present request for which the proxy is attempting to identify a matching cache entry. For example, the look-up engine **205** may look for a cache entry with an identifier which differs from the request identifier by a query parameter value. In utilizing this strategy, the look-up engine **205** can collect information collected for multiple previous requests (e.g., a list of arbitrary parameters in an identifier) to be later checked with the detected arbitrary parameter in the current request. For example, in the case where cache entries are stored with URI or URL identifiers, the look-up engine searches for a cache entry with a URI differing by a query parameter. If found, the engine **205** can examine the cache entry for information collected during previous requests (e.g. a list of arbitrary parameters) and checked whether the arbitrary parameter detected in or extracted from the current URI/URL belongs to the arbitrary parameters list.

Example 1.

Cache contains entry for <URL>/products/?query=all, where query is marked as arbitrary.

Request is being made to <URL>/products/?query=sub

Match will be found, since query parameter is marked as arbitrary.

Example 2.

Cache contains entry for <URL>/products/?query=all, where query is marked as arbitrary.

Request is being made to <URL>/products/?query=sub&sort=asc

Match will not be found, since current request contains sort parameter which is not marked as arbitrary in the cache entry.

Additional strategies for detecting cache hit may be employed. These strategies can be implemented singly or in any combination thereof. A cache-hit can be determined when any one of these strategies determines a match. A cache miss may be indicated when the look-up engine **205** determines that the requested data cannot be served from the cache **285**, for any reason. For example, a cache miss may be determined when no cache entries are identified for any or all utilized look-up strategies.

Cache miss may also be determined when a matching cache entry exists but determined to be invalid or irrelevant for the current request. For example, the look-up engine **205** may further analyze metadata (e.g., which may include timing data of the cache entry) associated with the matching cache entry to determine whether it is still suitable for use in responding to the present request.

When the look-up engine **205** has identified a cache hit (e.g., an event indicating that the requested data can be served from the cache), the stored response in the matching cache entry can be served from the cache to satisfy the request of an application/client.

By servicing requests using cache entries stored in cache **285**, network bandwidth and other resources need not be used to request/receive poll responses which may have not changed from a response that has already been received at the mobile device **250**. Such servicing and fulfilling application (e.g., mobile application) requests locally via cache entries in the local cache **285** allows for more efficient resource and mobile network traffic utilization and management since the request need not be sent over the wireless network further consuming bandwidth. In general, the cache **285** can be persisted between power on/off of the mobile device **250**, and persisted across application/client refreshes and restarts.

For example, the local proxy **275**, upon receipt of an outgoing request from its mobile device **250** or from an application or other type of client on the mobile device **250**, can intercept the request and determine whether a cached response is available in the local cache **285** of the mobile device **250**. If so, the outgoing request is responded to by the local proxy **275** using the cached response on the cache of the mobile device. As such, the outgoing request can be filled or satisfied without a need to send the outgoing request over the wireless network, thus conserving network resources and battery consumption.

In one embodiment, the responding to the requesting application/client on the device **250** is timed to correspond to a manner in which the content server would have responded to the outgoing request over a persistent connection (e.g., over the persistent connection, or long-held HTTP connection, long poll type connection, that would have been established absent interception by the local proxy). The timing of the response can be emulated or simulated by the local proxy **275** to preserve application behavior such that end user experience is not affected, or minimally affected by serving stored content from the local cache **285** rather than fresh content received from the intended content source (e.g., content host/application server **110** of FIG. **1**B-FIG. **1**C). The timing can be replicated exactly or estimated within a tolerance parameter, which may go unnoticed by the user or treated similarly by the application so as to not cause operation issues.

For example, the outgoing request can be a request for a persistent connection intended for the content server (e.g., application server/content provider of examples of FIGS. **1**B-**1**C). In a persistent connection (e.g., long poll, COMET-style push or any other push simulation in asynchronous HTTP requests, long-held HTTP request, HTTP streaming, or others) with a content source (server), the connection is held for some time after a request is sent. The connection can typically be persisted between the mobile device and the server until content is available at the server to be sent to the mobile device. Thus, there typically can be some delay in time between when a long poll request is sent and when a response is received from the content source. If a response is not provided by the content source for a certain amount of time, the connection may also terminate due to network reasons (e.g., socket closure) if a response is not sent.

Thus, to emulate a response from a content server sent over a persistent connection (e.g., a long poll style connection), the manner of response of the content server can be simulated by allowing a time interval to elapse before responding to the outgoing request with the cached response. The length of the time interval can be determined on a request by request basis or on an application by application (client by client basis), for example.

In one embodiment, the time interval is determined based on request characteristics (e.g., timing characteristics) of an application on the mobile device from which the outgoing request originates. For example, poll request intervals (e.g., which can be tracked, detected, and determined by the long poll detector **238***a* of the poll interval detector **238**) can be used to determine the time interval to wait before responding to a request with a local cache entry and managed by the response scheduler **249***a*.

One embodiment of the cache policy manager **245** includes a poll schedule generator **247** which can generate a polling schedule for one or more applications on the mobile device **250**. The polling schedule can specify a polling interval that can be employed by an entity which is physically distinct and/or separate from the mobile device **250** in monitoring the content source for one or more applications (such that cached responses can be verified periodically by polling a host server (host server **110** or **310**) to which the request is directed) on behalf of the mobile device. One example of such an external entity which can monitor the content at the source for the mobile device **250** is a proxy server (e.g., proxy server **125** or **325** shown in the examples of FIGS. **1**B-**1**C and FIG. **3**A-C).

The polling schedule (e.g., including a rate/frequency of polling) can be determined, for example, based on the interval between the polling requests directed to the content source from the mobile device. The polling schedule or rate of polling may be determined at the mobile device **250** (by the local proxy). In one embodiment, the poll interval detector **238** of the application behavior detector **236** can monitor polling requests directed to a content source from the mobile device **250** in order to determine an interval between the polling requests made from any or all application (e.g., mobile application).

For example, the poll interval detector **238** can track requests and responses for applications or clients on the device **250**. In one embodiment, consecutive requests are tracked prior to detection of an outgoing request initiated from the application (e.g., mobile application) on the mobile device **250** by the same mobile client or application (e.g., mobile application). The polling rate can be determined using request information collected for the request for which the response is cached. In one embodiment, the rate is determined from averages of time intervals between previous requests generated by the same client which generated the request. For example, a first interval may be computed between the current request and a previous request, and a second interval can be computed between the two previous requests. The polling rate can be set from the average of the first interval and the second interval and sent to the proxy server in setting up the caching strategy.

Alternate intervals may be computed in generating an average; for example, multiple previous requests in addition to two previous requests may be used, and more than two intervals may be used in computing an average. In general, in computing intervals, a given request need not have resulted in a response to be received from the host server/content source in order to use it for interval computation. In other words, the timing characteristics of a given request may be used in interval computation, as long as the request has been detected, even if the request failed in sending, or if the response retrieval failed.

One embodiment of the poll schedule generator **247** includes a schedule update engine **247***a* and/or a time adjustment engine **247***b*. The schedule update engine **247***a* can determine a need to update a rate or polling interval with which a given application server/content host from a previously set value, based on a detected interval change in the actual requests generated from a client or application (e.g., mobile application) on the mobile device **250**.

For example, a request for which a monitoring rate was determined may now be sent from the application (e.g., mobile application) or client at a different request interval. The scheduled update engine **247***a* can determine the updated polling interval of the actual requests and generate a new rate, different from the previously set rate to poll the host at on behalf of the mobile device **250**. The updated polling rate can be communicated to the remote proxy (proxy server **325**) over the cellular network for the remote proxy to monitor the given host. In some instances, the updated polling rate may be determined at the remote proxy or remote entity which monitors the host.

In one embodiment, the time adjustment engine **247***b* can further optimize the poll schedule generated to monitor the application server/content source (**110** or **310**). For example, the time adjustment engine **247***b* can optionally specify a time to start polling to the proxy server. For example, in addition to setting the polling interval at which the proxy server is to monitor the application, server/content host can also specify the time at which an actual request was generated at the mobile client/application.

However, in some cases, due to inherent transmission delay or added network delays or other types of latencies, the remote proxy server receives the poll setup from the local proxy with some delay (e.g., a few minutes, or a few seconds). This has the effect of detecting response change at the source after a request is generated by the mobile client/application causing the invalidate of the cached response to occur after it has once again been served to the application after the response is no longer current or valid.

To resolve this non-optimal result of serving the out-dated content once again before invalidating it, the time adjustment engine **247***b* can specify the time (t0) at which polling should begin in addition to the rate, where the specified initial time t0 can be specified to the proxy server **325** as a time that is less than the actual time when the request was generated by the mobile app/client. This way, the server polls the resource slightly before the generation of an actual request by the mobile client such that any content change can be detected prior to an actual application request. This prevents invalid or irrelevant out-dated content/response from being served once again before fresh content is served.

In one embodiment, an outgoing request from a mobile device **250** is detected to be for a persistent connection (e.g., a long poll, COMET style push, and long-held (HTTP) request) based on timing characteristics of prior requests from the same application or client on the mobile device **250**. For example, requests and/or corresponding responses can be tracked by the request/response tracking engine **238***b* of the long poll detector **238***a* of the poll interval detector **238**.

The timing characteristics of the consecutive requests can be determined to set up a polling schedule for the application or client. The polling schedule can be used to monitor the content source (content source/application server) for content changes such that cached content stored on the local cache in the mobile device **250** can be appropriately managed (e.g., updated or discarded). In one embodiment, the timing characteristics can include, for example, a response delay time ('D') and/or an idle time ('IT').

In one embodiment, the response/request tracking engine **238***b* can track requests and responses to determine, compute, and/or estimate, the timing diagrams for applicant or client requests.

For example, the response/request tracking engine **238***b* detects a first request (Request 0) initiated by a client on the mobile device and a second request (Request 1) initiated by the client on the mobile device after a response is received at the mobile device responsive to the first request. The second request is one that is subsequent to the first request.

In one embodiment, the response/request tracking engine **238***b* can track requests and responses to determine, compute, and/or estimate the timing diagrams for applicant or client requests. The response/request tracking engine **238***b* can detect a first request initiated by a client on the mobile device and a second request initiated by the client on the mobile device after a response is received at the mobile device responsive to the first request. The second request is one that is subsequent to the first request.

The response/request tracking engine **238***b* further determines relative timings between the first, second requests, and the response received in response to the first request. In general, the relative timings can be used by the long poll detector **238***a* to determine whether requests generated by the application are long poll requests.

Note that in general, the first and second requests that are used by the response/request tracking engine **238***b* in computing the relative timings are selected for use after a long poll hunting period has settled or in the event when long poll hunting does not occur. Timing characteristics that are typical of a long poll hunting period can be, for example, detected by the long poll hunting detector **238***c*. In other words, the requests tracked by the response/request tracking engine **238***b* and used for determining whether a given request is a long poll occurs after the long poll has settled.

In one embodiment, the long poll hunting detector **238***c* can identify or detect hunting mode, by identifying increasing request intervals (e.g., increasing delays). The long poll hunting detector **238***a* can also detect hunting mode by detecting increasing request intervals, followed by a request with no response (e.g., connection timed out), or by detecting increasing request intervals followed by a decrease in the interval. In addition, the long poll hunting detector **238***c* can apply a filter value or a threshold value to request-response time delay value (e.g., an absolute value) above which the detected delay can be considered to be a long poll request-response delay. The filter value can be any suitable value characteristic of long polls and/or network conditions (e.g., 2 s, 5 s, 10 s, 15 s, 20 s., etc.) and can be used as a filter or threshold value.

The response delay time ('D') refers to the start time to receive a response after a request has been sent and the idle refers to time to send a subsequent request after the response has been received. In one embodiment, the outgoing request is detected to be for a persistent connection based on a comparison (e.g., performed by the tracking engine **238***b*) of the response delay time relative ('D') or average of ('D') (e.g., any average over any period of time) to the idle time ('IT'), for example, by the long poll detector **238***a*. The number of averages used can be fixed, dynamically adjusted, or changed over a longer period of time. For example, the requests initiated by the client are determined to be long poll requests if the response delay time interval is greater than the idle time interval (D>IT or D>>IT). In one embodiment, the tracking engine **238***b* of the long poll detector computes, determines, or estimates the response delay time interval as the amount of time elapsed between time of the first request and initial detection or full receipt of the response.

In one embodiment, a request is detected to be for a persistent connection when the idle time ('IT') is short since persistent connections, established in response to long poll requests or long poll HTTP requests for example, can also be characterized in detecting immediate or near-immediate issuance of a subsequent request after receipt of a response to a previous request (e.g., IT ~0). As such, the idle time ('IT') can also be used to detect such immediate or near-immediate re-request to identify long poll requests. The absolute or relative timings determined by the tracking engine **238***b* are used to determine whether the second request is immediately or near-immediately re-requested after the response to the first request is received. For example, a request may be categorized as a long poll request if D+RT+IT~D+RT since IT is small for this to hold true. IT may be determined to be small if it is less than a threshold value. Note that the threshold value could be fixed or calculated over a limited time period (a session, a day, a month, etc.), or calculated over a longer time period (e.g., several months or the life of the analysis). For example, for every request, the average IT can be determined, and the threshold can be determined using this average IT (e.g., the average IT less a certain percentage may be used as the threshold). This can allow the threshold to automatically adapt over time to network conditions and changes in server capability, resource availability or server response. A fixed threshold can take upon any value including by way of example but not limitation (e.g., 1 s. 2 s. 3 s. . . . etc.).

In one embodiment, the long poll detector **238***a* can compare the relative timings (e.g., determined by the tracker engine **238***b*) to request-response timing characteristics for other applications to determine whether the requests of the application are long poll requests. For example, the requests initiated by a client or application can be determined to be long poll requests if the response delay interval time ('D') or the average response delay interval time (e.g., averaged over x number of requests or any number of delay interval times averaged over x amount of time) is greater than a threshold value.

The threshold value can be determined using response delay interval times for requests generated by other clients, for example by the request/response tracking engine **238***b* and/or by the application profile generator **239** (e.g., the response delay interval tracker **239***a*). The other clients may reside on the same mobile device and the threshold value is determined locally by components on the mobile device. The threshold value can be determined for all requests over all resources server over all networks, for example. The threshold value can be set to a specific constant value (e.g., 30 seconds, for example) to be used for all requests, or any request which does not have an applicable threshold value (e.g., long poll is detected if D>30 seconds).

In some instances, the other clients reside on different mobile devices and the threshold can be determined by a proxy server (e.g., proxy server **325** of the host **300** shown in the example of FIG. **3**A-B) which is external to the mobile device and able to communicate over a wireless network with the multiple different mobile devices, as will be further described with reference to FIG. **3**B.

In one embodiment, the cache policy manager **245** sends the polling schedule to the proxy server (e.g., proxy server **125** or **325** shown in the examples of FIGS. **1**B-**1**C and FIG.

3A) and can be used by the proxy server in monitoring the content source, for example, for changed or new content (updated response different from the cached response associated with a request or application). A polling schedule sent to the proxy can include multiple timing parameters including but not limited to interval (time from request 1 to request 2) or a time out interval (time to wait for response, used in long polls, for example). Referring to the timing diagram of a request/response timing sequence timing intervals 'RI', 'D', 'RT', and/or 'IT', or some statistical manipulation of the above values (e.g., average, standard deviation, etc.) may all or in part be sent to the proxy server.

For example, in the case when the local proxy **275** detects a long poll, the various timing intervals in a request/response timing sequence (e.g., 'D', 'RT', and/or 'IT') can be sent to the proxy server **325** for use in polling the content source (e.g., application server/content host **110**). The local proxy **275** can also identify to the proxy server **325** that a given application or request to be monitored is a long poll request (e.g., instructing the proxy server to set a 'long poll flag', for example). In addition, the proxy server uses the various timing intervals to determine when to send keep-alive indications on behalf of mobile devices.

The local cache invalidator **244** of the caching policy manager **245** can invalidate cache elements in the local cache (e.g., cache **185** or **285**) when new or changed data (e.g., updated response) is detected from the application server/content source for a given request. The cached response can be determined to be invalid for the outgoing request based on a notification received from the proxy server (e.g., proxy **325** or the host server **300**). The source which provides responses to requests of the mobile client can be monitored to determine relevancy of the cached response stored in the cache of the mobile device **250** for the request. For example, the cache invalidator **244** can further remove/delete the cached response from the cache of the mobile device when the cached response is no longer valid for a given request or a given application.

In one embodiment, the cached response is removed from the cache after it is provided once again to an application which generated the outgoing request after determining that the cached response is no longer valid. The cached response can be provided again without waiting for the time interval or provided again after waiting for a time interval (e.g., the time interval determined to be specific to emulate the response delay in a long poll). In one embodiment, the time interval is the response delay 'D' or an average value of the response delay 'D' over two or more values.

The new or changed data can be, for example, detected by the proxy server (e.g., proxy server **125** or **325** shown in the examples of FIGS. **1B-1C** and FIG. **3A**). When a cache entry for a given request/poll has been invalidated, the use of the radio on the mobile device **250** can be enabled (e.g., by the local proxy **275** or the cache policy manager **245**) to satisfy the subsequent polling requests, as further described with reference to the interaction diagram of FIG. **9-10**.

One embodiment of the cache policy manager **245** includes a cache or connect selection engine **249** which can decide whether to use a locally cached entry to satisfy a poll/content request generated at the mobile device **250** by an application or widget. For example, the local proxy **275** or the cache policy manger **245** can intercept a polling request, made by an application (e.g., mobile application) on the mobile device, to contact the application server/content provider. The selection engine **249** can determine whether the content received for the intercepted request has been locally stored as cache elements for deciding whether the radio of the mobile device needs to be activated to satisfy the request made by the application (e.g., mobile application) and also determine whether the cached response is still valid for the outgoing request prior to responding to the outgoing request using the cached response.

In one embodiment, the local proxy **275**, in response to determining that relevant cached content exists and is still valid, can retrieve the cached elements from the local cache to provide a response to the application (e.g., mobile application) which made the polling request such that a radio of the mobile device is not activated to provide the response to the application (e.g., mobile application). In general, the local proxy **275** continues to provide the cached response each time the outgoing request is received until the updated response different from the cached response is detected.

When it is determined that the cached response is no longer valid, a new request for a given request is transmitted over the wireless network for an updated response. The request can be transmitted to the application server/content provider (e.g., server/host **110**) or the proxy server on the host server (e.g., proxy **325** on the host **300**) for a new and updated response. In one embodiment the cached response can be provided again as a response to the outgoing request if a new response is not received within the time interval, prior to removal of the cached response from the cache on the mobile device.

FIG. **2C** depicts a block diagram illustrating another example of components in the application behavior detector **236** and the caching policy manager **245** in the local proxy **275** on the client-side of the distributed proxy system shown in the example of FIG. **2A**. The illustrated application behavior detector **236** and the caching policy manager **245** can, for example, enable the local proxy **275** to detect cache defeat and perform caching of content addressed by identifiers intended to defeat cache.

In one embodiment, the caching policy manager **245** includes a cache defeat resolution engine **221**, an identifier formalizer **211**, a cache appropriateness decision engine **246**, a poll schedule generator **247**, an application protocol module **248**, a cache or connect selection engine **249** having a cache query module **229**, and/or a local cache invalidator **244**. The cache defeat resolution engine **221** can further include a pattern extraction module **222** and/or a cache defeat parameter detector **223**. The cache defeat parameter detector **223** can further include a random parameter detector **224** and/or a time/date parameter detector **226**. One embodiment further includes an application cache policy repository **243** coupled to the decision engine **246**.

In one embodiment, the application behavior detector **236** includes a pattern detector **237**, a poll interval detector **238**, an application profile generator **239**, and/or a priority engine **241**. The pattern detector **237** can further include a cache defeat parameter detector **223** having also, for example, a random parameter detector **233** and/or a time/date parameter detector **234**. One embodiment further includes an application profile repository **242** coupled to the application profile generator **239**. The application profile generator **239**, and the priority engine **241** have been described in association with the description of the application behavior detector **236** in the example of FIG. **2A**.

The cache defeat resolution engine **221** can detect, identify, track, manage, and/or monitor content or content sources (e.g., servers or hosts) which employ identifiers and/or are addressed by identifiers (e.g., resource identifiers such as URLs and/or URIs) with one or more mechanisms that defeat cache or are intended to defeat cache. The cache defeat resolution engine **221** can, for example, detect from a given data request generated by an application or client that the identifier defeats or potentially defeats cache, where the data request otherwise addresses content or responses from a host or server (e.g., application server/content host **110** or **310**) that is cacheable.

In one embodiment, the cache defeat resolution engine **221** detects or identifies cache defeat mechanisms used by content sources (e.g., application server/content host **110** or **310**) using the identifier of a data request detected at the mobile device **250**. The cache defeat resolution engine **221** can detect or identify a parameter in the identifier which can indicate that cache defeat mechanism is used. For example, a format, syntax, or pattern of the parameter can be used to identify cache defeat (e.g., a pattern, format, or syntax as determined or extracted by the pattern extraction module **222**).

The pattern extraction module **222** can parse an identifier into multiple parameters or components and perform a matching algorithm on each parameter to identify any of which match one or more predetermined formats (e.g., a date and/or time format). For example, the results of the matching or the parsed out parameters from an identifier can be used (e.g., by the cache defeat parameter detector **223**) to identify cache defeating parameters which can include one or more changing parameters.

The cache defeat parameter detector **223**, in one embodiment can detect random parameters (e.g., by the random parameter detector **224**) and/or time and/or date parameters which are typically used for cache defeat. The cache defeat parameter detector **223** can detect random parameters and/or time/dates using commonly employed formats for these parameters and performing pattern matching algorithms and tests.

In addition to detecting patterns, formats, and/or syntaxes, the cache defeat parameter detector **223** further determines or confirms whether a given parameter is defeating cache and whether the addressed content can be cached by the distributed caching system. The cache defeat parameter detector **223** can detect this by analyzing responses received for the identifiers utilized by a given data request. In general, a changing parameter in the identifier is identified to indicate cache defeat when responses corresponding to multiple data requests are the same even when the multiple data requests uses identifiers with the changing parameter being different for each of the multiple data requests. For example, the request/response pairs illustrate that the responses received are the same, even though the resource identifier includes a parameter that changes with each request.

For example, at least two same responses may be required to identify the changing parameter as indicating cache defeat. In some instances, at least three same responses may be required. The requirement for the number of same responses needed to determine that a given parameter with a varying value between requests is cache defeating may be application specific, context dependent, and/or user dependent/user specified, or a combination of the above. Such a requirement may also be static or dynamically adjusted by the distributed cache system to meet certain performance thresholds and/or either explicit/implicit feedback regarding user experience (e.g., whether the user or application is receiving relevant/fresh content responsive to requests). More of the same responses may be required to confirm cache defeat, or for the system to treat a given parameter as intended for cache defeat if an application begins to malfunction due to response caching and/or if the user expresses dissatisfaction (explicit user feedback) or the system detects user frustration (implicit user cues).

The cache appropriateness decision engine **246** can detect, assess, or determine whether content from a content source (e.g., application server/content provider **110** in the example of FIG. **1**C) with which a mobile device **250** interacts, has content that may be suitable for caching. In some instances, content from a given application server/content provider (e.g., the server/provider **110** of FIG. **1**C) is determined to be suitable for caching based on a set of criteria (for example, criteria specifying time criticality of the content that is being requested from the content source). In one embodiment, the local proxy (e.g., the local proxy **175** or **275** of FIGS. **1**B-**1**C and FIG. **2**A) applies a selection criteria to store the content from the host server which is requested by an application as cached elements in a local cache on the mobile device to satisfy subsequent requests made by the application.

The selection criteria can also include, by way of example, but not limitation, state of the mobile device indicating whether the mobile device is active or inactive, network conditions, and/or radio coverage statistics. The cache appropriateness decision engine **246** can any one or any combination of the criteria, and in any order, in identifying sources for which caching may be suitable.

Once application servers/content providers having identified or detected content that is potentially suitable for local caching on the mobile device **250**, the cache policy manager **245** can proceed to cache the associated content received from the identified sources by storing content received from the content source as cache elements in a local cache (e.g., local cache **185** or **285** shown in the examples of FIGS. **1**B-**1**C and FIG. **2**A, respectively) on the mobile device **250**. The content source can also be identified to a proxy server (e.g., proxy server **125** or **325** shown in the examples of FIGS. **1**B-**1**C and FIG. **3**A, respectively) remote from and in wireless communication with the mobile device **250** such that the proxy server can monitor the content source (e.g., application server/content provider **110**) for new or changed data. Similarly, the local proxy (e.g., the local proxy **175** or **275** of FIGS. **1**B-**1**C and FIG. **2**A, respectively) can identify to the proxy server that content received from a specific application server/content provider is being stored as cached elements in the local cache.

In one embodiment, cache elements are stored in the local cache **285** as being associated with a normalized version of an identifier for an identifier employing one or more parameters intended to defeat cache. The identifier can be normalized by the identifier normalizer module **211** and the normalization process can include, by way of example, one or more of: converting the URI scheme and host to lower-case, capitalizing letters in percent-encoded escape sequences, removing a default port, and removing duplicate slashes.

In another embodiment, the identifier is normalized by removing the parameter for cache defeat and/or replacing the parameter with a static value which can be used to address or be associated with the cached response received responsive to a request utilizing the identifier by the normalizer **211** or the cache defeat parameter handler **212**. For example, the cached elements stored in the local cache **285** (shown in FIG. **2**A) can be identified using the normalized version of the identifier or a hash value of the normalized version of the identifier. The hash value of an identifier or of the normalized identifier may be generated by the hash engine **213**.

Once content has been locally cached, the cache policy manager **245** can, upon receiving future polling requests to contact the content server, retrieve the cached elements from the local cache to respond to the polling request made at the mobile device **250** such that a radio of the mobile device is not activated to service the polling request. Such servicing and fulfilling application (e.g., mobile application) requests locally via local cache entries allow for more efficient resource and mobile network traffic utilization and management since network bandwidth and other resources need not be used to request/receive poll responses which may have not changed from a response that has already been received at the mobile device **250**.

One embodiment of the cache policy manager **245** includes a poll schedule generator **247** which can generate a polling schedule for one or more applications on the mobile device **250**. The polling schedule can specify a polling interval that can be employed by the proxy server (e.g., proxy server **125** or **325** shown in the examples of FIGS. **1**B-**1**C and FIG. **3**A) in monitoring the content source for one or more applications. The polling schedule can be determined, for example, based on the interval between the polling requests directed to the content source from the mobile device. In one embodiment, the poll interval detector **238** of the application behavior detector can monitor polling requests directed to a content source from the mobile device **250** in order to determine an interval between the polling requests made from any or all application (e.g., mobile application).

In one embodiment, the cache policy manager **245** sends the polling schedule is sent to the proxy server (e.g., proxy server **125** or **325** shown in the examples of FIGS. **1**B-**1**C and FIG. **3**A) and can be used by the proxy server in monitoring the content source, for example, for changed or new content. The local cache invalidator **244** of the caching policy manager **245** can invalidate cache elements in the local cache (e.g., cache **185** or **285**) when new or changed data is detected from the application server/content source for a given request. The new or changed data can be, for example, detected by the proxy server. When a cache entry for a given request/poll has been invalidated and/or removed (e.g., deleted from cache) after invalidation, the use of the radio on the mobile device **250** can be enabled (e.g., by the local proxy or the cache policy manager **245**) to satisfy the subsequent polling requests, as further described with reference to the interaction diagram of FIG. **4**B.

In another embodiment, the proxy server (e.g., proxy server **125** or **325** shown in the examples of FIGS. **1**B-**1**C **1**C and FIG. **3**A) uses a modified version of a resource identifier used in a data request to monitor a given content source (the application server/content host **110** of FIGS. **1**B-**1**C to which the data request is addressed) for new or changed data. For example, in the instance where the content source or identifier is detected to employ cache defeat mechanisms, a modified (e.g., normalized) identifier can be used instead to poll the content source. The modified or normalized version of the identifier can be communicated to the proxy server by the caching policy manager **245**, or more specifically the cache defeat parameter handler **212** of the identifier normalizer **211**.

The modified identifier used by the proxy server to poll the content source on behalf of the mobile device/application (e.g., mobile application) may or may not be the same as the normalized identifier. For example, the normalized identifier may be the original identifier with the changing cache defeating parameter removed whereas the modified identifier uses a substitute parameter in place of the parameter that is used to defeat cache (e.g., the changing parameter replaced with a static value or other predetermined value known to the local proxy and/or proxy server). The modified parameter can be determined by the local proxy **275** and communicated to the proxy server. The modified parameter may also be generated by the proxy server (e.g., by the identifier modifier module **353** shown in the example of FIG. **3**C).

One embodiment of the cache policy manager **245** includes a cache or connect selection engine **249** which can decide whether to use a locally cached entry to satisfy a poll/content request generated at the mobile device **250** by an application or widget. For example, the local proxy **275** or the cache policy manger **245** can intercept a polling request made by an application (e.g., mobile application) on the mobile device, to contact the application server/content provider. The selection engine **249** can determine whether the content received for the intercepted request has been locally stored as cache elements for deciding whether the a radio of the mobile device needs to be activated to satisfy the request made by the application (e.g., mobile application). In one embodiment, the local proxy **275**, in response to determining that relevant cached content exists and is still valid, can retrieve the cached elements from the local cache to provide a response to the application (e.g., mobile application) which made the polling request such that a radio of the mobile device is not activated to provide the response to the application (e.g., mobile application).

In one embodiment, the cached elements stored in the local cache **285** (shown in FIG. **2**A) can be identified using a normalized version of the identifier or a hash value of the normalized version of the identifier, for example, using the cache query module **229**. Cached elements can be stored with normalized identifiers which have cache defeating parameters removed or otherwise replaced such that the relevant cached elements can be identified and retrieved in the future to satisfy other requests employing the same type of cache defeat. For example, when an identifier utilized in a subsequent request is determined to be utilizing the same cache defeating parameter, the normalized version of this identifier can be generated and used to identify a cached response stored in the mobile device cache to satisfy the data request. The hash value of an identifier or of the normalized identifier may be generated by the hash engine **213** of the identifier normalizer **211**.

FIG. **2**D depicts a block diagram illustrating examples of additional components in the local proxy **275** shown in the example of FIG. **2**A which is further capable of performing mobile traffic categorization and policy implementation based on application behavior and/or user activity.

In this embodiment of the local proxy **275**, the user activity module **215** further includes one or more of, a user activity tracker **215***a*, a user activity prediction engine **215***b*, and/or a user expectation manager **215***c*. The application behavior detect **236** can further include a prioritization engine **241***a*, a time criticality detection engine **241***b*, an application state categorizer **241***c*, and/or an application traffic categorizer **241***d*. The local proxy **275** can further include a backlight detector **219** and/or a network configuration selection engine **251**. The network configuration selection engine **251** can further include, one or more of, a wireless generation standard selector **251***a*, a data rate specifier **251***b*, an access channel selection engine **251***c*, and/or an access point selector **251***d*.

In one embodiment, the application behavior detector **236** is able to detect, determined, identify, or infer, the activity state of an application on the mobile device **250** to which traffic has originated from or is directed to, for example, via the application state categorizer **241***c* and/or the traffic categorizer **241***d*. The activity state can be determined by whether the application is in a foreground or background state on the mobile device (via the application state categorizer **241***c*) since the traffic for a foreground application vs. a background application may be handled differently.

In one embodiment, the activity state can be determined, detected, identified, or inferred with a level of certainty of heuristics, based on the backlight status of the mobile device **250** (e.g., by the backlight detector **219**) or other software agents or hardware sensors on the mobile device, including but not limited to, resistive sensors, capacitive sensors, ambient light sensors, motion sensors, touch sensors, etc. In general, if the backlight is on, the traffic can be treated as being or determined to be generated from an application that is active or in the foreground, or the traffic is interactive. In addition, if the backlight is on, the traffic can be treated as being or determined to be traffic from user interaction or user activity, or traffic containing data that the user is expecting within some time frame.

In one embodiment, the activity state is determined based on whether the traffic is interactive traffic or maintenance traffic. Interactive traffic can include transactions from responses and requests generated directly from user activity/interaction with an application and can include content or data that a user is waiting or expecting to receive. Maintenance traffic may be used to support the functionality of an application which is not directly detected by a user. Maintenance traffic can also include actions or transactions that may take place in response to a user action, but the user is not actively waiting for or expecting a response.

For example, a mail or message delete action at a mobile device **250** generates a request to delete the corresponding mail or message at the server, but the user typically is not waiting for a response. Thus, such a request may be categorized as maintenance traffic, or traffic having a lower priority (e.g., by the prioritization engine **241***a*) and/or is not time-critical (e.g., by the time criticality detection engine **214***b*).

Contrastingly, a mail 'read' or message 'read' request initiated by a user a the mobile device **250**, can be categorized as 'interactive traffic' since the user generally is waiting to access content or data when they request to read a message or mail. Similarly, such a request can be categorized as having higher priority (e.g., by the prioritization engine **241***a*) and/or as being time critical/time sensitive (e.g., by the time criticality detection engine **241***b*).

The time criticality detection engine **241***b* can generally determine, identify, infer the time sensitivity of data contained in traffic sent from the mobile device **250** or to the mobile device from a host server (e.g., host **300**) or application server (e.g., app server/content source **110**). For example, time sensitive data can include, status updates, stock information updates, IM presence information, email messages or other messages, actions generated from mobile gaming applications, webpage requests, location updates, etc. Data that is not time sensitive or time critical, by nature of the content or request, can include requests to delete messages, mark-as-read or edited actions, application-specific actions such as a add-friend or delete-friend request, certain types of messages, or other information which does not frequently changing by nature, etc. In some instances when the data is not time critical, the timing with which to allow the traffic to pass through is set based on when additional data needs to be sent from the mobile device **250**. For example, traffic shaping engine **255** can align the traffic with one or more subsequent transactions to be sent together in a single power-on event of the mobile device radio (e.g., using the alignment module **256** and/or the batching module **257**). The alignment module **256** can also align polling requests occurring close in time directed to the same host server, since these request are likely to be responded to with the same data.

In the alternate or in combination, the activity state can be determined from assessing, determining, evaluating, inferring, identifying user activity at the mobile device **250** (e.g., via the user activity module **215**). For example, user activity can be directly detected and tracked using the user activity tracker **215***a*. The traffic resulting therefrom can then be categorized appropriately for subsequent processing to determine the policy for handling. Furthermore, user activity can be predicted or anticipated by the user activity prediction engine **215***b*. By predicting user activity or anticipating user activity, the traffic thus occurring after the prediction can be treated as resulting from user activity and categorized appropriately to determine the transmission policy.

In addition, the user activity module **215** can also manage user expectations (e.g., via the user expectation manager **215***c* and/or in conjunction with the activity tracker **215** and/or the prediction engine **215***b*) to ensure that traffic is categorized appropriately such that user expectations are generally met. For example, a user-initiated action should be analyzed (e.g., by the expectation manager **215**) to determine or infer whether the user would be waiting for a response. If so, such traffic should be handled under a policy such that the user does not experience an unpleasant delay in receiving such a response or action.

In one embodiment, an advanced generation wireless standard network is selected for use in sending traffic between a mobile device and a host server in the wireless network based on the activity state of the application on the mobile device for which traffic is originated from or directed to. An advanced technology standards such as the 3G, 3.5G, 3G+, 4G, or LTE network can be selected for handling traffic generated as a result of user interaction, user activity, or traffic containing data that the user is expecting or waiting for. Advanced generation wireless standard network can also be selected for to transmit data contained in traffic directed to the mobile device which responds to foreground activities.

In categorizing traffic and defining a transmission policy for mobile traffic, a network configuration can be selected for use (e.g., by the network configuration selection engine **251**) on the mobile device **250** in sending traffic between the mobile device and a proxy server (**325**) and/or an application server (e.g., app server/host **110**). The network configuration that is selected can be determined based on information gathered by the application behavior module **236** regarding application activity state (e.g., background or foreground traffic), application traffic category (e.g., interactive or maintenance traffic), any priorities of the data/content, time sensitivity/criticality.

The network configuration selection engine **2510** can select or specify one or more of, a generation standard (e.g., via wireless generation standard selector **251***a*), a data rate (e.g., via data rate specifier **251***b*), an access channel (e.g., access channel selection engine **251***c*), and/or an access point (e.g., via the access point selector **251***d*), in any combination.

For example, a more advanced generation (e.g., 3G, LTE, or 4G or later) can be selected or specified for traffic when the activity state is in interaction with a user or in a foreground on the mobile device. Contrastingly, an older generation standard (e.g., 2G, 2.5G, or 3G or older) can be specified for traffic when one or more of the following is detected, the application is not interacting with the user, the application is running in the background on the mobile device, or the data contained in the traffic is not time critical, or is otherwise determined to have lower priority.

Similarly, a network configuration with a slower data rate can be specified for traffic when one or more of the following is detected, the application is not interacting with the user, the application is running in the background on the mobile device, or the data contained in the traffic is not time critical. The access channel (e.g., Forward access channel or dedicated channel) can be specified.

FIG. **3**A depicts a block diagram illustrating an example of server-side components in a distributed proxy and cache system residing on a host server **300** that manages traffic in a wireless network for resource conservation. The server-side proxy (or proxy server **325**) can further categorize mobile traffic and/or implement delivery policies based on application behavior, content priority, user activity, and/or user expectations.

The host server **300** generally includes, for example, a network interface **308** and/or one or more repositories **312**, **314**, and **316**. Note that server **300** may be any portable/mobile or non-portable device, server, cluster of computers and/or other types of processing units (e.g., any number of a machine shown in the example of FIG. **16**) able to receive or transmit signals to satisfy data requests over a network including any wired or wireless networks (e.g., WiFi, cellular, Bluetooth, etc.).

The network interface **308** can include networking module(s) or devices(s) that enable the server **300** to mediate data in a network with an entity that is external to the host server **300**, through any known and/or convenient communications protocol supported by the host and the external entity. Specifically, the network interface **308** allows the server **300** to communicate with multiple devices including mobile phone devices **350** and/or one or more application servers/content providers **310**.

The host server **300** can store information about connections (e.g., network characteristics, conditions, types of connections, etc.) with devices in the connection metadata repository **312**. Additionally, any information about third party application or content providers can also be stored in the repository **312**. The host server **300** can store information about devices (e.g., hardware capability, properties, device settings, device language, network capability, manufacturer, device model, OS, OS version, etc.) in the device information repository **314**. Additionally, the host server **300** can store information about network providers and the various network service areas in the network service provider repository **316**.

The communication enabled by network interface **308** allows for simultaneous connections (e.g., including cellular connections) with devices **350** and/or connections (e.g., including wired/wireless, HTTP, Internet connections, LAN, WiFi, etc.) with content servers/providers **310** to manage the traffic between devices **350** and content providers **310** or other servers such as an ad server **320***a*, promotional content server **320***b*, or an e-coupon server **320***c* for optimizing network resource utilization and/or to conserve power (battery) consumption on the serviced devices **350**. The host server **300** can communicate with mobile devices **350** serviced by different network service providers and/or in the same/different network service areas. The host server **300** can operate and is compatible with devices **350** with varying types or levels of mobile capabilities, including by way of example but not limitation, 1G, 2G, 2G transitional (2.5G, 2.75G), 3G (IMT-2000), 3G transitional (3.5G, 3.75G, 3.9G), 4G (IMT-advanced), etc.

In general, the network interface **308** can include one or more of a network adaptor card, a wireless network interface card (e.g., SMS interface, WiFi interface, interfaces for various generations of mobile communication standards including but not limited to 1G, 2G, 3G, 3.5G, 4G type networks such as LTE, WiMAX, etc.), Bluetooth, WiFi, or any other network whether or not connected via a router, an access point, a wireless router, a switch, a multilayer switch, a protocol converter, a gateway, a bridge, a bridge router, a hub, a digital media receiver, and/or a repeater.

The host server **300** can further include server-side components of the distributed proxy and cache system which can include a proxy server **325** and a server cache **335**. In one embodiment, the proxy server **325** can include an HTTP access engine **345**, a caching policy manager **355**, a proxy controller **365**, a traffic shaping engine **375**, a new data detector **347** and/or a connection manager **395**.

The HTTP access engine **345** may further include a heartbeat manager **398**; the proxy controller **365** may further include a data invalidator module **368**; the traffic shaping engine **375** may further include a control protocol **376** and a batching module **377**. Additional or less components/modules/engines can be included in the proxy server **325** and each illustrated component.

As used herein, a "module," a "manager," a "handler," a "detector," an "interface," a "controller," a "normalizer," a "generator," an "invalidator," or an "engine" includes a general purpose, dedicated or shared processor and, typically, firmware or software modules that are executed by the processor. Depending upon implementation-specific or other considerations, the module, manager, handler, detector, interface, controller, normalizer, generator, invalidator, or engine can be centralized or its functionality distributed. The module, manager, handler, detector, interface, controller, normalizer, generator, invalidator, or engine can include general or special purpose hardware, firmware, or software embodied in a computer-readable (storage) medium for execution by the processor. As used herein, a computer-readable medium or computer-readable storage medium is intended to include all mediums that are statutory (e.g., in the United States, under 35 U.S.C. 101), and to specifically exclude all mediums that are non-statutory in nature to the extent that the exclusion is necessary for a claim that includes the computer-readable (storage) medium to be valid. Known statutory computer-readable mediums include hardware (e.g., registers, random access memory (RAM), non-volatile (NV) storage, to name a few), but may or may not be limited to hardware.

In the example of a device (e.g., mobile device **350**) making an application or content request to an application server or content provider **310**, the request may be intercepted and routed to the proxy server **325** which is coupled to the device **350** and the application server/content provider **310**. Specifically, the proxy server is able to communicate with the local proxy (e.g., proxy **175** and **275** of the examples of FIG. **1** and FIG. **2** respectively) of the mobile device **350**, the local proxy forwards the data request to the proxy server **325** in some instances for further processing and, if needed, for transmission to the application server/content server **310** for a response to the data request.

In such a configuration, the host **300**, or the proxy server **325** in the host server **300** can utilize intelligent information provided by the local proxy in adjusting its communication with the device in such a manner that optimizes use of network and device resources. For example, the proxy server **325** can identify characteristics of user activity on the device **350** to modify its communication frequency. The characteristics of user activity can be determined by, for example, the activity/behavior awareness module **366** in the proxy controller **365** via information collected by the local proxy on the device **350**.

In one embodiment, communication frequency can be controlled by the connection manager **395** of the proxy server **325**, for example, to adjust push frequency of content or updates to the device **350**. For instance, push frequency can be decreased by the connection manager **395** when characteristics of the user activity indicate that the user is inactive. In one embodiment, when the characteristics of the user activity indicate that the user is subsequently active after a period of inactivity, the connection manager **395** can adjust the communication frequency with the device **350** to send data that was buffered as a result of decreased communication frequency to the device **350**.

In addition, the proxy server **325** includes priority awareness of various requests, transactions, sessions, applications, and/or specific events. Such awareness can be determined by the local proxy on the device **350** and provided to the proxy server **325**. The priority awareness module **367** of the proxy server **325** can generally assess the priority (e.g., including time-criticality, time-sensitivity, etc.) of various events or applications; additionally, the priority awareness module **367** can track priorities determined by local proxies of devices **350**.

In one embodiment, through priority awareness, the connection manager **395** can further modify communication frequency (e.g., use or radio as controlled by the radio controller **396**) of the server **300** with the devices **350**. For example, the server **300** can notify the device **350**, thus requesting use of the radio if it is not already in use when data or updates of an importance/priority level which meets a criteria becomes available to be sent.

In one embodiment, the proxy server **325** can detect multiple occurrences of events (e.g., transactions, content, data received from server/provider **310**) and allow the events to accumulate for batch transfer to device **350**. Batch transfer can be cumulated and transfer of events can be delayed based on priority awareness and/or user activity/application behavior awareness as tracked by modules **367** and/or **366**. For example, batch transfer of multiple events (of a lower priority) to the device **350** can be initiated by the batching module **377** when an event of a higher priority (meeting a threshold or criteria) is detected at the server **300**. In addition, batch transfer from the server **300** can be triggered when the server receives data from the device **350**, indicating that the device radio is already in use and is thus on. In one embodiment, the proxy server **325** can order the each messages/packets in a batch for transmission based on event/transaction priority such that higher priority content can be sent first in case connection is lost or the battery dies, etc.

In one embodiment, the server **300** caches data (e.g., as managed by the caching policy manager **355**) such that communication frequency over a network (e.g., cellular network) with the device **350** can be modified (e.g., decreased). The data can be cached, for example, in the server cache **335** for subsequent retrieval or batch sending to the device **350** to potentially decrease the need to turn on the device **350** radio. The server cache **335** can be partially or wholly internal to the host server **300**, although in the example of FIG. **3**A it is shown as being external to the host **300**. In some instances, the server cache **335** may be the same as and/or integrated in part or in whole with another cache managed by another entity (e.g., the optional caching proxy server **199** shown in the example of FIG. **1**C), such as being managed by an application server/content provider **310**, a network service provider, or another third party.

In one embodiment, content caching is performed locally on the device **350** with the assistance of host server **300**. For example, proxy server **325** in the host server **300** can query the application server/provider **310** with requests and monitor changes in responses. When changed or new responses are detected (e.g., by the new data detector **347**), the proxy server **325** can notify the mobile device **350** such that the local proxy on the device **350** can make the decision to invalidate (e.g., indicated as out-dated) the relevant cache entries stored as any responses in its local cache. Alternatively, the data invalidator module **368** can automatically instruct the local proxy of the device **350** to invalidate certain cached data, based on received responses from the application server/provider **310**. The cached data is marked as invalid, and can get replaced or deleted when new content is received from the content server **310**.

Note that data change can be detected by the detector **347** in one or more ways. For example, the server/provider **310** can notify the host server **300** upon a change. The change can also be detected at the host server **300** in response to a direct poll of the source server/provider **310**. In some instances, the proxy server **325** can in addition, pre-load the local cache on the device **350** with the new/updated data. This can be performed when the host server **300** detects that the radio on the mobile device is already in use, or when the server **300** has additional content/data to be sent to the device **350**.

One or more the above mechanisms can be implemented simultaneously or adjusted/configured based on application (e.g., different policies for different servers/providers **310**). In some instances, the source provider/server **310** may notify the host **300** for certain types of events (e.g., events meeting a priority threshold level). In addition, the provider/server **310** may be configured to notify the host **300** at specific time intervals, regardless of event priority.

In one embodiment, the proxy server **325** of the host **300** can monitor/track responses received for the data request from the content source for changed results prior to returning the result to the mobile device, such monitoring may be suitable when data request to the content source has yielded same results to be returned to the mobile device, thus preventing network/power consumption from being used when no new changes are made to a particular requested. The local proxy of the device **350** can instruct the proxy server **325** to perform such monitoring or the proxy server **325** can automatically initiate such a process upon receiving a certain number of the same responses (e.g., or a number of the same responses in a period of time) for a particular request.

In one embodiment, the server **300**, through the activity/behavior awareness module **366**, is able to identify or detect user activity at a device that is separate from the mobile device **350**. For example, the module **366** may detect that a user's message inbox (e.g., email or types of inbox) is being accessed. This can indicate that the user is interacting with his/her application using a device other than the mobile device **350** and may not need frequent updates, if at all.

The server **300**, in this instance, can thus decrease the frequency with which new or updated content is sent to the mobile device **350**, or eliminate all communication for as long as the user is detected to be using another device for access. Such frequency decrease may be application specific (e.g., for the application with which the user is interacting with on another device), or it may be a general frequency decrease (E.g., since the user is detected to be interacting with one server or one application via another device, he/she could also use it to access other services.) to the mobile device **350**.

In one embodiment, the host server **300** is able to poll content sources **310** on behalf of devices **350** to conserve power or battery consumption on devices **350**. For example, certain applications on the mobile device **350** can poll its respective server **310** in a predictable recurring fashion. Such recurrence or other types of application behaviors can be tracked by the activity/behavior module **366** in the proxy controller **365**. The host server **300** can thus poll content sources **310** for applications on the mobile device **350** that would otherwise be performed by the device **350** through a wireless (e.g., including cellular connectivity). The host server can poll the sources **310** for new or changed data by way of the HTTP access engine **345** to establish HTTP connection or by way of radio controller **396** to connect to the source **310** over the cellular network. When new or changed data is detected, the new data detector **347** can notify the device **350** that such data is available and/or provide the new/changed data to the device **350**.

In one embodiment, the connection manager **395** determines that the mobile device **350** is unavailable (e.g., the radio is turned off) and utilizes SMS to transmit content to the device **350**, for instance, via the SMSC shown in the example of FIG. **1**C. SMS is used to transmit invalidation messages, batches of invalidation messages, or even content in the case where the content is small enough to fit into just a few (usually one or two) SMS messages. This avoids the need to access the radio channel to send overhead information. The host server **300** can use SMS for certain transactions or responses having a priority level above a threshold or otherwise meeting a criteria. The server **300** can also utilize SMS as an out-of-band trigger to maintain or wake-up an IP connection as an alternative to maintaining an always-on IP connection. In one embodiment, connection manager **395** may include an Internet/WiFi controller **397** for this purpose.

In one embodiment, the connection manager **395** in the proxy server **325** (e.g., the heartbeat manager **398**) can generate and/or transmit heartbeat messages on behalf of connected devices **350** to maintain a backend connection with a provider **310** for applications running on devices **350**.

For example, in the distributed proxy system, local cache on the device **350** can prevent any or all heartbeat messages needed to maintain TCP/IP connections required for applications from being sent over the cellular, or other, network and instead rely on the proxy server **325** on the host server **300** to generate and/or send the heartbeat messages to maintain a connection with the backend (e.g., application server/provider **110** in the example of FIG. **1**A). The proxy server can generate the keep-alive (heartbeat) messages independent of the operations of the local proxy on the mobile device.

The repositories **312**, **314**, and/or **316** can additionally store software, descriptive data, images, system information, drivers, and/or any other data item utilized by other components of the host server **300** and/or any other servers for operation. The repositories may be managed by a database management system (DBMS), for example, which may be but is not limited to Oracle, DB2, Microsoft Access, Microsoft SQL Server, PostgreSQL, MySQL, FileMaker, etc.

The repositories can be implemented via object-oriented technology and/or via text files and can be managed by a distributed database management system, an object-oriented database management system (OODBMS) (e.g., ConceptBase, FastDB Main Memory Database Management System, JDOInstruments, ObjectDB, etc.), an object-relational database management system (ORDBMS) (e.g., Informix, OpenLink Virtuoso, VMDS, etc.), a file system, and/or any other convenient or known database management package.

FIG. **3**B depicts a block diagram illustrating a further example of components in the caching policy manager **355** in the cache system shown in the example of FIG. **3**A which is capable of caching and adapting caching strategies for application (e.g., mobile application) behavior and/or network conditions.

The caching policy manager **355**, in one embodiment, can further include a metadata generator **303**, a cache look-up engine **305**, an application protocol module **356**, a content source monitoring engine **357** having a poll schedule manager **358**, a response analyzer **361**, and/or an updated or new content detector **359**. In one embodiment, the poll schedule manager **358** further includes a host timing simulator **358***a*, a long poll request detector/manager **358***b*, a schedule update engine **358***c*, and/or a time adjustment engine **358***d*. The metadata generator **303** and/or the cache look-up engine **305** can be coupled to the cache **335** (or, server cache) for modification or addition to cache entries or querying thereof.

In one embodiment, the proxy server (e.g., the proxy server **125** or **325** of the examples of FIGS. **1**B-**1**C and FIG. **3**A) can monitor a content source for new or changed data via the monitoring engine **357**. The proxy server, as shown, is an entity external to the mobile device **250** of FIG. **2**A-B. The content source (e.g., application server/content provider **110** of FIGS. **1**B-**1**C) can be one that has been identified to the proxy server (e.g., by the local proxy) as having content that is being locally cached on a mobile device (e.g., mobile device **150** or **250**). The content source can be monitored, for example, by the monitoring engine **357** at a frequency that is based on polling frequency of the content source at the mobile device. The poll schedule can be generated, for example, by the local proxy and sent to the proxy server. The poll frequency can be tracked and/or managed by the poll schedule manager **358**.

For example, the proxy server can poll the host (e.g., content provider/application server) on behalf of the mobile device and simulate the polling behavior of the client to the host via the host timing simulator **358***a*. The polling behavior can be simulated to include characteristics of a long poll request-response sequences experienced in a persistent connection with the host (e.g., by the long poll request detector/manager **358***b*). Note that once a polling interval/behavior is set, the local proxy **275** on the device-side and/or the proxy server **325** on the server-side can verify whether application and application server/content host behavior match or can be represented by this predicted pattern. In general, the local proxy and/or the proxy server can detect deviations and, when appropriate, re-evaluate and compute, determine, or estimate another polling interval.

In one embodiment, the caching policy manager **355** on the server-side of the distribute proxy can, in conjunction with or independent of the proxy server **275** on the mobile device, identify or detect long poll requests. For example, the caching policy manager **355** can determine a threshold value to be used in comparison with a response delay interval time in a request-response sequence for an application request to identify or detect long poll requests, possible long poll requests (e.g., requests for a persistent connection with a host with which the client communicates including, but not limited to, a long-held HTTP request, a persistent connection enabling COMET style push, request for HTTP streaming, etc.), or other requests which can otherwise be treated as a long poll request.

For example, the threshold value can be determined by the proxy **325** using response delay interval times for requests generated by clients/applications across mobile devices which may be serviced by multiple different cellular or wireless networks. Since the proxy **325** resides on host **300** is able to communicate with multiple mobile devices via multiple networks, the caching policy manager **355** has access to application/client information at a global level which can be used in setting threshold values to categorize and detect long polls.

By tracking response delay interval times across applications across devices over different or same networks, the caching policy manager **355** can set one or more threshold values to be used in comparison with response delay interval times for long poll detection. Threshold values set by the proxy server **325** can be static or dynamic, and can be associated with conditions and/or a time-to-live (an expiration time/date in relative or absolute terms).

In addition, the caching policy manager **355** of the proxy **325** can further determine the threshold value, in whole or in part, based on network delays of a given wireless network, networks serviced by a given carrier (service provider), or multiple wireless networks. The proxy **325** can also determine the threshold value for identification of long poll requests based on delays of one or more application server/content provider (e.g., **110**) to which application (e.g., mobile application) or mobile client requests are directed.

The proxy server can detect new or changed data at a monitored content source and transmits a message to the mobile device notifying it of such a change such that the mobile device (or the local proxy on the mobile device) can take appropriate action (e.g., to invalidate the cache elements in the local cache). In some instances, the proxy server (e.g., the caching policy manager **355**) upon detecting new or changed data can also store the new or changed data in its cache (e.g., the server cache **135** or **335** of the examples of FIG. **1**C and FIG. **3**A, respectively). The new/updated data stored in the server cache **335** can be used in some instances to satisfy content requests at the mobile device; for example, it can be used after the proxy server has notified the mobile device of the new/changed content and that the locally cached content has been invalidated.

The metadata generator **303**, similar to the metadata generator **203** shown in the example of FIG. **2**B, can generate metadata for responses cached for requests at the mobile device **250**. The metadata generator **303** can generate metadata for cache entries stored in the server cache **335**. Similarly, the cache look-up engine **305** can include the same or similar functions are those described for the cache look-up engine **205** shown in the example of FIG. **2**B.

The response analyzer **361** can perform any or all of the functionalities related to analyzing responses received for requests generated at the mobile device **250** in the same or similar fashion to the response analyzer **246***d* of the local proxy shown in the example of FIG. **2**B. Since the proxy server **325** is able to receive responses from the application server/content source **310** directed to the mobile device **250**, the proxy server **325** (e.g., the response analyzer **361**) can perform similar response analysis steps to determine cacheability, as described for the response analyzer of the local proxy. The responses can be analyzed in addition to or in lieu of the analysis that can be performed at the local proxy **275** on the mobile device **250**.

Furthermore, the schedule update engine **358***c* can update the polling interval of a given application server/content host based on application request interval changes of the application at the mobile device **250** as described for the schedule update engine in the local proxy **275**. The time adjustment engine **358***d* can set an initial time at which polls of the application server/content host is to begin to prevent the serving of out of date content once again before serving fresh content as described for the schedule update engine in the local proxy **275**. Both the schedule updating and the time adjustment algorithms can be performed in conjunction with or in lieu of the similar processes performed at the local proxy **275** on the mobile device **250**.

FIG. **3**C depicts a block diagram illustrating another example of components in the caching policy manager **355** in the proxy server **375** on the server-side of the distributed proxy system shown in the example of FIG. **3**A which is capable of managing and detecting cache defeating mechanisms and monitoring content sources.

The caching policy manager **355**, in one embodiment, can further include a cache defeating source manager **352**, a content source monitoring engine **357** having a poll schedule manager **358**, and/or an updated or new content detector **359**. The cache defeating source manager **352** can further include an identifier modifier module **353** and/or an identifier pattern tracking module **354**.

In one embodiment, the proxy server (e.g., the proxy server **125** or **325** of the examples of FIGS. **1**B-**1**C and FIG. **3**A) can monitor a content source for new or changed data via the monitoring engine **357**. The content source (e.g., application server/content provider **110** of FIGS. **1**B-**1**C or **310** of FIG. **3**A) can be one that has been identified to the proxy server (e.g., by the local proxy) as having content that is being locally cached on a mobile device (e.g., mobile device **150** or **250**). The content source **310** can be monitored, for example, by the monitoring engine **357** at a frequency that is based on polling frequency of the content source at the mobile device. The poll schedule can be generated, for example, by the local proxy and sent to the proxy server **325**. The poll frequency can be tracked and/or managed by the poll schedule manager **358**.

In one embodiment, the proxy server **325** uses a normalized identifier or modified identifier in polling the content source **310** to detect new or changed data (responses). The normalized identifier or modified identifier can also be used by the proxy server **325** in storing responses on the server cache **335**. In general, the normalized or modified identifiers can be used when cache defeat mechanisms are employed for cacheable content. Cache defeat mechanisms can be in the form of a changing parameter in an identifier such as a URI or URL and can include a changing time/data parameter, a randomly varying parameter, or other types parameters.

The normalized identifier or modified identifier removes or otherwise replaces the changing parameter for association with subsequent requests and identification of associated responses and can also be used to poll the content source. In one embodiment, the modified identifier is generated by the cache defeating source manager **352** (e.g., the identifier modifier module **353**) of the caching policy manager **355** on the proxy server **325** (server-side component of the distributed proxy system). The modified identifier can utilize a substitute parameter (which is generally static over a period of time) in place of the changing parameter that is used to defeat cache.

The cache defeating source manager **352** optionally includes the identifier pattern tracking module **354** to track, store, and monitor the various modifications of an identifier or identifiers that address content for one or more content sources (e.g., application server/content host **110** or **310**) to continuously verify that the modified identifiers and/or normalized identifiers used by the proxy server **325** to poll the content sources work as predicted or intended (e.g., receive the same responses or responses that are otherwise still relevant compared to the original, unmodified identifier).

In the event that the pattern tracking module **354** detects a modification or normalization of an identifier that causes erratic or unpredictable behavior (e.g., unexpected responses to be sent) on the content source, the tracking module **354** can log the modification and instruct the cache defeating source manager **352** to generate another modification/normalization, or notify the local proxy (e.g., local proxy **275**) to generate another modification/normalization for use in polling the content source. In the alternative or in parallel, the requests from the given mobile application/client on the mobile device (e.g., mobile device **250**) can temporarily be sent across the network to the content source for direct responses to be provided to the mobile device and/or until a modification of an identifier which works can be generated.

In one embodiment, responses are stored as server cache elements in the server cache when new or changed data is detected for a response that is already stored on a local cache (e.g., cache **285**) of the mobile device (e.g., mobile device **250**). Therefore, the mobile device or local proxy **275** can connect to the proxy server **325** to retrieve the new or changed data for a response to a request which was previously cached locally in the local cache **285** (now invalid, out-dated, or otherwise determined to be irrelevant).

The proxy server **325** can detect new or changed data at a monitored application server/content host **310** and transmits a message to the mobile device notifying it of such a change such that the mobile device (or the local proxy on the mobile device) can take appropriate action (e.g., to invalidate the cache elements in the local cache). In some instances, the proxy server (e.g., the caching policy manager **355**), upon detecting new or changed data, can also store the new or changed data in its cache (e.g., the server cache **135** or **335** of the examples of FIG. **1**C and FIG. **3**A, respectively). The updated/new data stored in the server cache can be used, in some instances, to satisfy content requests at the mobile device; for example, it can be used after the proxy server has notified the mobile device of the new/changed content and that the locally cached content has been invalidated.

FIG. **3**D depicts a block diagram illustrating examples of additional components in proxy server **325** shown in the example of FIG. **3**A which is further capable of performing mobile traffic categorization and policy implementation based on application behavior and/or traffic priority.

In one embodiment of the proxy server **325**, the traffic shaping engine **375** is further coupled to a traffic analyzer **336** for categorizing mobile traffic for policy definition and implementation for mobile traffic and transactions directed to one or more mobile devices (e.g., mobile device **250** of FIG. **2**A-**2**D) or to an application server/content host (e.g., **110** of FIGS. **1**B-**1**C). In general, the proxy server **325** is remote from the mobile devices and remote from the host server, as shown in the examples of FIGS. **1**B-**1**C. The proxy server **325** or the host server **300** can monitor the traffic for multiple mobile devices and is capable of categorizing traffic and devising traffic policies for different mobile devices.

In addition, the proxy server **325** or host server **300** can operate with multiple carriers or network operators and can implement carrier-specific policies relating to categorization of traffic and implementation of traffic policies for the various categories. For example, the traffic analyzer **336** of the proxy server **325** or host server **300** can include one or more of, a prioritization engine **341***a*, a time criticality detection engine **341***b*, an application state categorizer **341***c*, and/or an application traffic categorizer **341***d*.

Each of these engines or modules can track different criterion for what is considered priority, time critical, background/foreground, or interactive/maintenance based on different wireless carriers. Different criterion may also exist for different mobile device types (e.g., device model, manufacturer, operating system, etc.). In some instances, the user of the mobile devices can adjust the settings or criterion regarding traffic category and the proxy server **325** is able to track and implement these user adjusted/configured settings.

In one embodiment, the traffic analyzer **336** is able to detect, determined, identify, or infer, the activity state of an application on one or more mobile devices (e.g., mobile device **150** or **250**) which traffic has originated from or is directed to, for example, via the application state categorizer **341***c* and/or the traffic categorizer **341***d*. The activity state can be determined based on whether the application is in a foreground or background state on one or more of the mobile devices (via the application state categorizer **341***c*) since the traffic for a foreground application vs. a background application may be handled differently to optimize network use.

In the alternate or in combination, the activity state of an application can be determined by the wirelessly connected mobile devices (e.g., via the application behavior detectors in the local proxies) and communicated to the proxy server **325**. For example, the activity state can be determined, detected, identified, or inferred with a level of certainty of heuristics, based on the backlight status at mobile devices (e.g., by a backlight detector) or other software agents or hardware sensors on the mobile device, including but not limited to, resistive sensors, capacitive sensors, ambient light sensors, motion sensors, touch sensors, etc. In general, if the backlight is on, the traffic can be treated as being or determined to be generated from an application that is active or in the foreground, or the traffic is interactive. In addition, if the backlight is on, the traffic can be treated as being or determined to be traffic from user interaction or user activity, or traffic containing data that the user is expecting within some time frame.

The activity state can be determined from assessing, determining, evaluating, inferring, identifying user activity at the mobile device **250** (e.g., via the user activity module **215**) and communicated to the proxy server **325**. In one embodiment, the activity state is determined based on whether the traffic is interactive traffic or maintenance traffic. Interactive traffic can include transactions from responses and requests generated directly from user activity/interaction with an application and can include content or data that a user is waiting or expecting to receive. Maintenance traffic may be used to support the functionality of an application which is not directly detected by a user. Maintenance traffic can also include actions or transactions that may take place in response to a user action, but the user is not actively waiting for or expecting a response.

The time criticality detection engine **341***b* can generally determine, identify, infer the time sensitivity of data contained in traffic sent from the mobile device **250** or to the mobile device from the host server **300** or proxy server **325**, or the application server (e.g., app server/content source **110**). For example, time sensitive data can include, status updates, stock information updates, IM presence information, email messages or other messages, actions generated from mobile gaming applications, webpage requests, location updates, etc.

Data that is not time sensitive or time critical, by nature of the content or request, can include requests to delete messages, mark-as-read or edited actions, application-specific actions such as a add-friend or delete-friend request, certain types of messages, or other information which does not frequently changing by nature, etc. In some instances when the data is not time critical, the timing with which to allow the traffic to be sent to a mobile device is based on when there is additional data that needs to the sent to the same mobile device. For example, traffic shaping engine **375** can align the traffic with one or more subsequent transactions to be sent together in a single power-on event of the mobile device radio (e.g., using the alignment module **378** and/or the batching module **377**). The alignment module **378** can also align polling requests occurring close in time directed to the same host server, since these request are likely to be responded to with the same data.

In general, whether new or changed data is sent from a host server to a mobile device can be determined based on whether an application on the mobile device to which the new or changed data is relevant, is running in a foreground (e.g., by the application state categorizer **341***c*), or the priority or time criticality of the new or changed data. The proxy server **325** can send the new or changed data to the mobile device if the application is in the foreground on the mobile device, or if the application is in the foreground and in an active state interacting with a user on the mobile device, and/or whether a user is waiting for a response that would be provided in the new or changed data. The proxy server **325** (or traffic shaping engine **375**) can send the new or changed data that is of a high priority or is time critical.

Similarly, the proxy server **325** (or the traffic shaping engine **375**) can suppressing the sending of the new or changed data if the application is in the background on the mobile device. The proxy server **325** can also suppress the sending of the new or changed data if the user is not waiting for the response provided in the new or changed data; wherein the suppressing is performed by a proxy server coupled to the host server and able to wirelessly connect to the mobile device.

In general, if data, including new or change data is of a low priority or is not time critical, the proxy server can waiting to transfer the data until after a time period, or until there is additional data to be sent (e.g. via the alignment module **378** and/or the batching module **377**).

Client-Side Proxy

It is noted that, in the following, certain acronyms are used for convenience. Their functional descriptions are introduced throughout this disclosure as well as documents cited herein.

| Acronym | Meaning |
|---|---|
| RR | Responses to Requests |
| D | Delay |
| RMP | Rapid Manual Poll |
| RLP | Rapid Long Poll |
| RI | Request Interval |
| IT | Idle Time |
| RT | Response Time |
| LP | Long Poll |

FIG. **4**A depicts a block diagram illustrating another example of client-side components in a distributed proxy and cache system, further including an extended caching optimization engine.

FIG. **4**B depicts a block diagram illustrating additional components in the extended caching optimization engine shown in the example of FIG. **4**A.

It is noted that the functionalities of these modules **402**-**406** may be included, either partially or wholly, in the one or more modules introduced in FIGS. **2**A-**2**D. For example, the optimization engine **401** can be merged with, work as a portion of, communicate with, supplement, or cooperate with the request/transaction manager **235**, the caching policy manager **245**, the traffic shaping engine **255**, the connection manager **265**, etc.

In accordance with one or more embodiments, the extended caching optimization (ECO) engine **401** can detect or determine various external, internal, or derived factors using various detection, monitor, and pattern extraction modules **402**-**408**. Examples of these factors may include: (1) determination of user inactivity (e.g., based on screen state, motion sensors, and so forth; (2) determination of radio availability; (3) prediction of user activity based on previous patterns; or (4) network health state (e.g. congestion).

Based on these factors (and/or based on different setting levels of the configurable ECO settings, as mentioned earlier), the optimization engine **401** can determine (or take more chances in) whether it should continue to rely on one or more particular data stored in the local cache **285**. In some embodiments, the determination can be probabilistic instead of deterministic.

More specifically, to decide whether to refresh one or more data in the cache **285**, a cache freshness determination module **420** can determine based on at least one or more of (1) the ECO setting level **410**, (2) the actual freshness of the data, and (3) the device specific information including, for example, whether the user is using the device, how important the polling application is, whether the device is low in battery or in a congested network, or whether the user is driving, etc.

For purposes of discussion herein, "continuing to rely on cache" means that the ECO algorithm (e.g., as implemented in the optimization engine **401**) continues to consider the data stored in the cache are fresh and useable (a) when the determination of freshness is unknown (e.g., where the server that is monitoring the freshness is unable to retrieve the resource to determine the freshness—but content may be fresh), or (b) when the ECO algorithm knows that the resource is not fresh, but when situational factors listed above allow relaxing the strict freshness requirements and taking more chances as either user is not believed to need the data, or network resources need to be preserved.

Furthermore, in some embodiments, the optimization engine **402** allows caching proxies (e.g., local proxy **105**, FIG. **1**A; client-side proxy **175**, FIG. **1**E) to serve data that is not known to be fresh (or in some cases, known to be not fresh). This technique can be applied to (a) resources cached (e.g., in local cache **285**) by the polling logics (not shown in FIG. **4**B for simplicity) and (b) situations where user/network factors justify such behavior.

In one example, in order to determine the actual freshness of a response (e.g., whether a data stored in local cache **285** is fresh or stale), the optimization engine **401** compares its freshness lifetime to its age to determine if the response has expired. These may be implemented by incorporating protocols and calculations as specified in the HTTP/1.1 family of standards. One specific example of expiration calculation is specified in the section "13.2 Expiration Model" of the Request for Comment (RFC) #2616 or 2068 (also known as the "RFC 2616" or "RFC 2068"). The calculations of age, freshness, and expiration can be employed by the optimization engine **401** as a factor or a basis (in addition to the above mentioned factors including, for example, user activity, radio state, mobile application characteristics, network status, battery status, display/backlit state, etc) in determining whether to poll or refresh content stored in the local cache **285**.

Example Functions of Extended Caching and Behaviors Under Different Setting Levels

In general, extended caching (which can be also referred to herein as aggressive caching) techniques enable a system (e.g., mobile device **101**, **250** employing local proxy **105**, **275**, and/or host host server **111**, **300** employing proxy server **113**, **325** (see also FIG. **5**A-**5**B)) to take more risk in keeping providing application with the cached content in various situations described below.

Extended Caching: HTTP Non-Periodic Request

As part of normal operation, the Signaling Optimization client (e.g., local proxy **105**) caches resources on the device (e.g., in cache **185**) based on detecting a periodic application request (such as via application behavior detector **236**, described above). By default, such cached resources are served from cache only in response to application requests that match the observed periodicity. Open Channel Signaling Optimization identifies the opportunity to cache content based on observation of recurring request-response patterns. That is to say, request periodicity typically serves as a basis of request-response pattern recognition. Both simple and complex periodicities are detected.

However, the present embodiments recognize that applications can make "out-of-order" requests, and therefore the present embodiments include an HTTP Non-periodic-request option which provides conditions upon which the client serves cached resources in response to application requests that are non-periodic.

More specifically, when application makes a request "out-of-order" in context of the current polling pattern, local proxy **105** and/or proxy server **113** do not know whether the content has changed or not; however, given that periodic polling is taking place, the ECO engine **107** and/or ECO manager **115** can make the proxies **105**, **113** take the risk by not polling the third-party servers **119** for new data/update. By default, local proxy **105** and/or proxy server **113** take this risk when the screen is off or even when screen is off but the radio is inactive as this is rather low risk.

In some embodiments, configuration parameters can be adapted to adjust this. An example of behaviors under different setting levels the present embodiments perform is listed as follows:

| Setting | Behavior |
|---|---|
| 0 | Default behavior. Non-periodic requests are not served from cache. |
| 1 | Non-periodic requests ARE served from cache when the screen is not lit. |
| 2 | Non-periodic requests ARE served from cache when the screen is not lit OR the radio is inactive. |
| 3 | Non-periodic requests are always served from cache. |

For another example, the present embodiments recognize that aggressive caching, especially responding to out-of-order requests, can lead to application getting into a loop making requests immediately when a response from cache is provided, and such loop can go on infinitely and spiral out of control. As such, the present disclosed embodiments are designed to safeguard against such bad behavior

It is noted that, for convenience, a client (e.g., local local proxy **105**, **175**, **275**) of the distributed caching system can be referred to herein as an "open channel client", or "OC client." Similarly, a server (e.g., host server **111**, **100**, **300** hosting proxy server **113**, **125**, **325**) of the distributed caching system can be referred to herein as an "open channel server", or "OC server." The client and the server individually or together implementing the distributed caching techniques (including the Signal Optimization and Extended Caching techniques) can be referred to as "open channel" or "OC."

In some additional or alternative embodiments, as the aggressive pattern recognition above leads to recognizing long polls that just happen to be getting a response from the server early (e.g. a new email) as regular polls (to which an OC client responds immediately), it caused conditions where application makes a request, the OC client responds immediately, and application makes immediately a new request, and this loop continues. Without aggressive caching, this would only take place for very limited period of time, as the OC server (e.g., proxy server **113**, **325**), while polling for the resource, recognizes that the origin server (e.g., third-party server **119**A) no longer provides this response, sends an invalidate to the client. Or, if the recognized period is very long and server is polling at such long interval (e.g. hours), and has not observed the change yet, the requests from the application would have been considered out-of-order and passed to the network. However, now combined with aggressive caching, this loop can continue for hours, until user turns the screen on, and the invalidate is processed (or the request is considered out-of-order).

Accordingly, the present embodiments include a safeguard module **440** which includes safeguard functionalities to reduce or solve this issue. Module **440** includes two submodules **440**A and **440**B for detecting this looping behavior:

Starting to increase the response delay when immediate same requests are observed, to avoid battery drain (Rapid-poll—Battery Drain Safeguard)

Deactivate the aggressive caching feature to process the invalidate and deactivate serving out-of-order requests from cache (Temporarily Deactivation)

Rapid-poll—Battery Drain Safeguard: One specific example is "Exchange Activesync," or "EAS." Without current techniques, rapid polls and battery drain can be caused by the distributed caching system. For example, when OC would detect RMP (as the delays look more like network latency, and the observed RIs are too large for an RLP), and OC starts serving from cache immediately and EAS immediately polls again, thereby getting the system in a loop and drains the battery.

Accordingly, the ECO engine **401** includes a Rapid-poll Battery Drain Safeguard module **440**A that provides the aforementioned distributed caching system with a rapid poll safeguard mechanism. Safeguard module **440** detects rapidly caching Responses to Requests (RR) (from the application polling) and starts delaying responses from the cache, improving the battery life. This can be applicable to Rapid Manual Poll (RMP), Rapid Long Poll (RLP) and Request Interval (RI) polling classes.

These classes are different patterns for pattern recognition. RMP represents normal poll with request interval shorter than a threshold (e.g., 60 seconds)), and RLP represents long poll, with response delay shorter than a threshold (e.g., 60 seconds)). More details regarding these classes are further introduced in, for example, U.S. patent application Ser. No. 13/274,265, entitled "CACHING ADAPTED FOR MOBILE APPLICATION BEHAVIOR AND NETWORK CONDITIONS," filed Oct. 14, 2011, which is incorporated herein by reference in its entirety.

Additionally, the necessity for safeguard behavior can be detected based on specified condition, which can be described as "a hit that causes new request".

For example, when arithmetic average of IT (idle time) between two subsequent "hit" requests is smaller than certain number (e.g., 15 sec), OC starts serving responses from the cache with increased delay D with a default step specified in code (e.g., 10 sec). OC tracks the application if the delayed response is received successfully and the application did not close the socket (e.g., no IN socket error and app works correctly). The IN SOCKET closure typically happens when: a) the application crash; b) the application don't accept the delay.

In case the socket was not closed, OC keeps increasing D until max value (1.5*Dmin_for_LP) is reached. If the originally detected pattern was RI and max D was reached, OC detects Long Poll (LP) and updates information about the polling pattern on the server. If the originally detected pattern was RMP or RLP, OC keeps HITting with max D until pattern expiration. In case IN socket error was observed, D is decreased with a step interval. If the socket was closed two times in a row, OC cancels safeguard mechanism for this poll. If the condition for safeguard mechanism is met again, OC starts it over.

Temporary Deactivation: the present embodiments further recognize cases where aggressive caching led to OC Client ignoring the resource invalidation from the OC Server, while the client application went into a loop of requesting the resource continuously.

Accordingly, the ECO engine **401** includes a Temporary Deactivation module **440**B that detects this condition and can temporarily deactivate aggressive caching for the given RR. Any outstanding invalidate-with-cache or invalidate-without-cache (e.g., invalidation commands sent from OC server invalidating data stored in OC client's cache, and further indicating whether the OC server has the new data) has to be applied immediately upon entering the temporary deactivation mode. The aggressiveness should be re-activated on a consequent startpoll.

In addition to process outstanding invalidates, out-of-order requests are to be passed to the network after entering the temporary deactivation mode. This, in some embodiments, can be re-activated not only on a subsequent startpoll, but also if safeguard is deactivated.

Extended Caching: Screen State Change

In some embodiments, in the optimization engine **401**'s determination of whether the screen backlight state has changed, a grace period is included that defines for how long the screen has to be off before the state is considered to have changed. The grace period setting can be a configurable parameter. For example, a screen has to be off for X seconds before the state is considered to have changed when the setting is at the lowest level, and for Y seconds when the setting is at the highest level.

Extended Caching: Domain Name Service (DNS) Cache Entry (CE) Expiration

The present embodiments recognize that, from a practical standpoint, it is often hard to predict or know what application issues a DNS request (e.g., especially when the request gets hidden when traversing through a certain OS system such as Android), it would be desirable not to have the DNS requests turn on the radio while the firewall (e.g., as configured by OC) blocking the actual application request, thereby resulting in little resource saving because the radio is already turned up due to the DNS request.

For example, for some particular applications (e.g., WhatsApp messenger application by WhatsApp Inc.), even though the WhatApp application may be blocked by personal firewall rule as implemented by OC client, the application is still able to send DNS request (e.g., to "c1.whatsapp.net") and blocked only when it was trying to connect to that server.

Accordingly, the ECO engine **401** includes a DNS CE Expiration module **450** that caches DNS requests more aggressively to prevent unnecessary radio turn-ons triggered by those DNS requests. In particular, besides being based on when an DNS Cache Lookup Query (CLQ) is made to OC client and on a typical time-to-live (TTL) parameter, the DNS CE Expiration module **450** can adjust the expiration of DNS cache entries stored in the cache based on the ECO level settings and/or device specific information.

An example policy setting for controlling the client's behavior when invalidating DNS cache entries due to their expiration is listed below.

| Setting | Behavior |
|---|---|
| 0 | regular behavior, DNS Cache Entry expire on DNS CLQ when the TTL ends; |
| 1 | DNS Cache Entry expire on DNS CLQ when the TTL ends and the screen backlight is on; |
| 2 | DNS Cache Entry expire on DNS CLQ when the TTL ends and the screen backlight is on and radio is up; |
| 3 | DNS Cache Entry never expires. |

Extended Caching: HTTP Invalidate-Without-Cache

As part of normal operation, the Signaling Optimization client (e.g., local proxy **105**) caches resources on the device (e.g., in cache **185**) and requests the Signaling Optimization server (e.g., proxy server **113**) to poll to ensure freshness of a cached resource. If the server determines that a cached resource has changed, the server may send an "invalidate-without-cache" message (e.g., via a corresponding module **560** in the optimization manager **501**) to the client. The invalidate-without-cache message notifies the client that the server is not able to verify the freshness of a cached resource (invalidate) and that the client should contact the application's origin server directly for the next update ("-without-cache"). The can happen when the OC server is not able to retrieve any content from the original, third-party server, and therefore it is hard to tell whether something has changed.

Accordingly, when server sends client an "invalidate w/o cache" message, the present embodiments can take more risk in keeping to provide application with the cached content.

More specifically, upon receiving an invalidate-without-cache message, the default client behavior then is to invalidate the cached resource immediately. The Extended HTTP Invalidate-without-cache setting, as implemented by an "invalidate-without-cache" module **460** that is included in the optimization engine **401**, provides options for the conditions upon which the client invalidates the cached resource.

An example policy setting for controlling the client's behavior when "invalidate-without-cache" the cache entries is listed below.

| Setting | Behavior |
|---|---|
| 0 | Default behavior. Cache entries are invalidated immediately upon receipt of cache-invalidate message. |

-continued

| Setting | Behavior |
|---|---|
| 1 | Cache entries based on periodic application polling are invalidated on cache lookup query with screen lit. Cache entries based on application long-polls are invalided on screen lit. |
| 2 | Cache entries based on periodic application polling are invalidated on cache lookup query with screen lit AND radio up. Cache entries based on application long-polls are invalided on screen lit AND radio up. |
| 3 | Invalidate-without-cache messages are ignored completely. |

It is noted that, with "long polling" (see discussion above), the client requests information from the server in a similar way to a normal poll. However, if the server does not have any information available for the client, instead of sending an empty response, the server holds the request and waits for some information to be available. Once the information becomes available (or after a suitable timeout), a complete response is sent to the client. The client will normally then immediately re-request information from the server, so that the server will almost always have an available waiting request that it can use to deliver data in response to an event.

Extended Caching: HTTP Invalidate-with-Cache

As part of normal operation, the Signaling Optimization client (e.g., local proxy **105**) caches resources on the device and requests the Signaling Optimization server (e.g., proxy server **113**) to poll to ensure freshness of a cached resource. If the Signaling Optimization server determines that a cached resource has been updated on the application server, the Signaling Optimization server may cache a copy of the updated resource locally and send an "invalidate-with-cache" message (e.g., via a corresponding module **570** in the optimization manager **501**) to the client. The invalidate-without-cache message notifies the client that the specified resource has changed (invalidate) and that the Signaling Optimization server has cached the changed resource ("-with- cache"). The Signaling Optimization client may choose to contact the Signaling Optimization server directly to retrieve the updated resource.

That is to say, when server sends "invalidate-with-cache" messages, it means that content as definitely changed, and the present embodiments can take more risk in keeping to provide application with the cached content.

More specifically, the present embodiments can delay processing of this invalidate when screen is off (but process when screen is on, even if radio is inactive), despite of knowing that the content has changed, because it is determined that user does not need the content as the user is not actively using the device (e.g., via device specific information module **430**).

As such, upon receiving an invalidate-with-cache message, the default client behavior is to retrieve the updated resource entry from the origin server upon the first occurrence entry of either the radio coming up or the application requesting the resource. The Extended HTTP Invalidate-with-cache setting, as implemented by an "invalidate-with-cache" module **470** that is included in the optimization engine **401**, provides options for the conditions upon which the Signaling Optimization client retrieves the remotely cached resource entry from the Signaling Optimization server.

An example policy setting for controlling the client's behavior when "invalidate-with-cache" the cache entries is listed below.

| Setting | Behavior |
|---|---|
| 0 | Default behavior. Remote cache entry is retrieved on either radio up OR application |
| 1 | Remote cache resources based on periodic application polling are retrieved on cache lookup query and screen lit. Remote cache resources based on application long-polls are retrieved on screen lit. |
| 2 | Remote cache resources based on periodic application polling are retrieved on cache lookup query and screen lit and radio up. Remote cache resources based on application long-polls are retrieved on screen lit and radio up. |
| 3 | Completely ignore invalidates. |

In some embodiments, all the ignored invalidates can be picked up as soon as the backlight turns on. Also, with the highest aggressiveness level, some embodiments can include an extra subscription state "already invalidated" which would be treated as "polling" while the aggressiveness level is at highest, but would lead to an immediate invalidate on a cache lookup query (CLQ) otherwise.

An 'extra subscription state' represents a subscription that relates to a cached entry and subscription by the client to server to poll for that cached entry to monitor its freshness. It is the new subscription type that tells that the cache entry has been invalidated, but the invalidate has not been processed yet due to the aggressiveness setting.

Irregular Polling Pattern Recognition and Caching

The present embodiments recognize that some applications are to poll in a non-periodic fashion, which can lead to OC client occasionally missing the requests and re-detecting increasing RI patterns with shorter RI. Once the poll invalidates, OC client tries re-detecting the pattern based on the recent request history, which leads to the same story repeating over and over.

Accordingly, in order to improve efficiency (among other purposes), an Irregular Polling Pattern Recognition and Caching (IPPRC) module **480** is included to persist the information on the shortest observed RI across the invalidates. The IPPRC module **480** can activate under circumstances of upgrading an existing increasing pattern to another increasing, with a shorter RI.

It is noted that, as previously mentioned, the Irregular Polling Pattern Recognition and Caching module **480** can be working as a part of the optimization **401**, can be working as a supplemental component of application behavior detector **236** (and/or submodules thereof such as the pattern detector **237**), and/or can be combined, merged, or separated from other suitable modules/components in the local proxy **275**.

More specifically, as aggressive caching typically only applies to requests that are already being cached, The IPPRC module **480** can detect patterns in any occasion where multiple requests occur, even if the pattern would violate the previous recognition rules where period **2** should be same or longer than period **1**.

In addition, IPPRC module **480**'s can provide functionality which relates to "semi-long polls", which is a pattern that are identified where the pattern fluctuates between normal polls and long polls. Typically, this would not be recognized as a pattern at all. With IPPRC module **480**, it would get recognized as a long poll.

For one example, if the long poll delay parameter sent by Exchange Activesync (EAS) doesn't change (and get thus normalized out), but the server just decides to respond sooner every second time, IPPRC module **480** could start caching at the longer interval, as the application should wait for that time, and it doesn't matter if the server responds faster, because the application would just issue a new poll and continue. In some embodiments, the IPPRC module **440** use max response delay from event history to detect LP pattern. In case of semi-long poll pattern behavior OC will start polling after second long D in history

To further demonstrate how the functionality works, example cases are provided below:

Test Cases

In general, to perform a test case, (1) OC client (e.g., in forms of software) must be existing (e.g., by installation) on the device; (2) a test tool (e.g., a proprietary test tool "7TestTool," as provided by SEVEN Networks, Inc.) should be installed; (3) A test resource is needed for this test case that returns the same response for all requests.

In general, to verify correct results, these steps should be performed: (1) Open 7TestTool application and load the test suite; (2) Start periodic request; and (3) Observe client log.

The following are the example test cases and the responses.

| Summary | Pattern/Delay | Result |
|---|---|---|
| | Regression Test Cases | |
| Detection of Rapid Manual Poll | 1. Install OC client<br>2. Install latest available 7TestTool.<br>3. A test resource is needed for this test case that returns the same response for all requests.<br>Pattern [0, 35, 35, 35] | 1. After 3rd request Rapid Manual Poll should be detected.<br>2. After receiving of 3rd response RR should be activated,<br>polling should start.<br>3. 4th request should be HITed with delay 0. |
| Detection of Rapid Long Poll | 1. Install OC client<br>2. Install latest available 7TestTool.<br>3. A test resource is needed for this test case that returns the same response for all requests.<br>Pattern [0, 35, 35, 35]<br>Delay [24, 21, 21, 21] | 1. After 3rd request Rapid Long Poll should be detected.<br>2. After receiving of 3rd response RR should be activated,<br>polling should start.<br>3. 4th request should be HITed with delay 20. |
| Detection of Long Poll | 1. Install OC client<br>2. Install latest available 7TestTool.<br>3. A test resource is needed for this test case that returns the same response for all requests.<br>Pattern [0, 70, 70]<br>Delay [68, 68, 68] | 1. After 2nd request Long poll pattern should be detected with next value getRecentTO: recent TO = 68<br>2. After receiving of 2nd response RR should be activated,<br>polling should start.<br>3. 3th request should be HITed with delay 68. |
| Detection of RI with delay | 1. Install OC client<br>2. Install latest available 7TestTool.<br>3. A test resource is needed for this test case that returns the same response for all requests.<br>Pattern [0, 70, 70, 70]<br>Delay [20, 10, 31] | 1. After 3rd request RI based pattern should be detected.<br>2. After receiving of 3rd response RR should be activated,<br>polling should start.<br>3. 4th request should be HITed with delay 0. |
| Detection of RI based polling | 1. Install OC client<br>2. Install latest available 7TestTool.<br>3. A test resource is needed for this test case that returns the same response for all requests.<br>Pattern [0, 65, 65, 65] | 1. After 3rd request RI based polling should be detected.<br>2. After receiving of 3rd response RR should be activated,<br>polling should start.<br>3. 4th request should be HITed with delay 0. |
| OC should redetect RR from Long Poll to RI | 1. Install OC client<br>2. Install latest available 7TestTool.<br>3. A test resource is needed for this test case that returns the same response for all requests.<br>Pattern [0, 69, 71, 70, 70]<br>Delay [66, 11, 3, 0] | 1. After 2nd request Long poll should be detected.<br>2. Response delay of 2nd response with value 11 doesn't match the current pattern.<br>3. 3rd request should be sent to TC for server side revalidation. After 3rd request RI based polling detected with interval: 68.<br>4. After receiving of 3rd response RR should be activated,<br>polling should start.<br>5. 4th response should be HITed with delay 0. |
| OC shouldn't redetect RR from Long Poll to RI | 1. Install OC client<br>2. Install latest available 7TestTool.<br>3. A test resource is needed for this test case that returns the same response for all requests.<br>Pattern [0, 65, 65, 65]<br>Delay [35, 35, 34] | 1. After 3rd request RI based polling should be detected.<br>2. After receiving of 3rd response RR should be activated,<br>polling should start.<br>3. 4th request should be HITed with delay 0. |

-continued

| Summary | Pattern/Delay | Result |
|---|---|---|
| OC should redetect RR from Rapid Long Poll to Long Poll | 1. Install OC client<br>2. Install latest available 7TestTool.<br>3. A test resource is needed for this test case that returns the same response for all requests.<br>Pattern<br>[0, 35, 35, 35, 65, 65, 65, 65]<br>Delay<br>[21, 21, 25, 25, 65, 65, 65, 65] | 1. After 3rd request Rapid Long Poll should be detected.<br>2. After receiving of 3rd response RR should be activated,<br>polling should start.<br>3. 4th request should be HITed with delay 25.<br>4. 5th request should be HITed with delay 25.<br>5. Before 6th request INVALIDATE_WO_CACHE should be received.<br>6. 6th request should be sent to TC for server side revalidation. RI should be detected with interval: 70.<br>7. 7th request should be sent to TC for server side revalidation. Long Poll should be detected.<br>8. After 7th response polling should start.<br>9. 8th request should be HITed. |
| | Functional Test Cases | |
| OC behavior for requests with pattern [0, 65, 100, 35, 45, 65, 70, 80, 45, 67, 65, 100, 35, 45, 65, 70, 80, 45, 67] | 1. Install OC client<br>2. Install latest available 7TestTool.<br>3. A test resource is needed for this test case that returns the same response for all requests.<br>Pattern<br>[0, 65, 100, 35, 45, 65, 70, 80, 45, 67, 65, 100, 35, 45, 65, 70, 80, 45, 67] | 1. After 3rd request RI should be detected with period 47.<br>2. After receiving of 3rd response RR should be activated,<br>polling should start with interval 60.<br>3. 4th request should be sent to TC for server side revalidation.<br>4. 5th request should be HITed.<br>5. 6th request should be HITed.<br>6. 7th request should be HITed.<br>7. 8th request should be HITed.<br>8. 9th request should be sent to TC for server side revalidation.<br>9. 10th request should be HITed.<br>10. 11th request should be HITed.<br>11. 12th request should be HITed.<br>12. 13th request should be sent to TC for server side revalidation.<br>13. 14th-17th request should be HITed.<br>14. 18th request should be sent to TC for server side revalidation.<br>15. 19th request should be HITed. |
| OC behavior for requests with pattern [0, 70, 2, 100, 35, 45, 10, 45, 35, 25, 67, 70, 2, 100, 35, 45, 10, 45, 35, 25, 67] | 1. Install OC client<br>2. Install latest available 7TestTool.<br>3. A test resource is needed for this test case that returns the same response for all requests.<br>Pattern<br>[0, 70, 2, 100, 35, 45, 10, 45, 35, 25, 67, 70, 2, 100, 35, 45, 10, 45, 35, 25, 67] | 1. After 3rd request RI should be detected with period 70.<br>2. After receiving of 3rd response RR should be activated,<br>polling should start with interval 70.<br>3. 4th request should be HITed.<br>4. 5th request should be sent to TC for server side revalidation. Polling should start with new interval 60.<br>5. 6th request should be HITed.<br>6. 7th request should be HITed due to it were sent in short period of time.<br>7. 8th request should be HITed.<br>8. 9th request should be sent to TC for server side revalidation, temporary RMP should start.<br>9. 10th request should be HITed.<br>10. 11th request should be HITed.<br>11. 12th request should be HITed.<br>12. 13th request should be HITed due to it were sent in short period of time.<br>13. 14th request should be HITed.<br>14. 15th request should be sent to TC for server side revalidation. RMP expired.<br>15. 16th request should be HITed.<br>16. 17th request should be HITed due to it were sent in short period of time.<br>17. 18th request should be HITed.<br>18. 19th request should be sent to TC for server side revalidation. Temporary RMP should start.<br>19. 20th request should be HITed.<br>20. 21 th request should be HITed. |

-continued

| Summary | Pattern/Delay | Result |
|---|---|---|
| OC behavior for requests with pattern [0, 155, 200, 80, 80, 200, 70, 70, 153, 13, 11, 170, 202, 155, 200, 80, 80, 200, 70, 70, 153, 13, 11, 170, 202] | 1. Install OC client<br>2. Install latest available 7TestTool.<br>3. A test resource is needed for this test case that returns the same response for all requests.<br>Pattern<br>[0, 155, 200, 80, 80, 200, 70, 70, 153, 13, 11, 170, 202, 155, 200, 80, 80, 200, 70, 70, 153, 13, 11, 170, 202] | 1. After 3rd request RI should be detected with period 132.<br>2. After receiving of 3rd response RR should be activated, polling should start.<br>3. 4th request should be HITed.<br>4. 5th request should be sent to TC for server side revalidation. Polling should start with new interval 60.<br>5. 6th request should be HITed.<br>6. 7th request should be HITed.<br>7. 8th request should be HITed.<br>8. 9th request should be HITed.<br>9. 10th request should be HITed due to it were sent in short period of time.<br>10. 11th request should be HITed due to it were sent in short period of time.<br>11. 12th-21th requests should be HITed.<br>12. 22th-23th requests should be HITed due to it were sent in short period of time.<br>13. 24th-25th requests should be HITed. |
| OC behavior for requests with pattern [0, 350, 325, 400, 323, 345, 360, 300, 398, 378, 350, 325, 400, 323, 345, 360, 300, 398, 378] | 1. Install OC client<br>2. Install latest available 7TestTool.<br>3. A test resource is needed for this test case that returns the same response for all requests.<br>Pattern<br>[0, 350, 325, 400, 323, 345, 360, 300, 398, 378, 350, 325, 400, 323, 345, 360, 300, 398, 378] | 1. After 3rd request RI should be detected with period 313.<br>2. After receiving of 3rd response RR should be activated, polling should start.<br>3. 4th-19th requests should be HITed. |
| OC behavior for requests with pattern [0, 300, 310, 296, 303, 299, 300, 308, 307, 301, 296, 299, 305, 301, 294, 302] | 1. Install OC client<br>2. Install latest available 7TestTool.<br>3. A test resource is needed for this test case that returns the same response for all requests.<br>Pattern<br>[0, 300, 310, 296, 303, 299, 300, 308, 307, 301, 296, 299, 305, 301, 294, 302] | 1. After 3rd request RI should be detected with period 295.<br>2. After receiving of 3rd response RR should be activated, polling should start.<br>3. 4th-16th requests should be HITed. |

The CLQ is a client-internal interface to check from the cache that what is the status of the subscription—to understand whether we have response for a given request in the cache, and is it ok to serve it to the application.

Heterogenous Cache Service and Categorization of Responses

The present embodiments recognize that certain requests/responses can be categorized (e.g., as “1-1-1-1-2-2-2-2” and “0-0-0-0-1-0-0-0” types), and they can be served with the response from cache based on their categorizations, even if polling is not ongoing, based on:

for one type (e.g., “1-1-1-1-2-2-2-2”) return the latest response from the network,

for another type (e.g., “0-0-0-0-1-0-0-0”) return the latest cached result.

Accordingly, the optimization engine **401** can include a heterogenous cache service module **490** that expand the scope of aggressive caching to situations where caching of a specific response has not taken place. For example, heterogenous cache service module **490** can categorize applications (e.g., based on their requests, and responses to those requests from corresponding third-party servers) and the data (e.g., responses) currently stored in the cache. Heterogenous cache service module **490** can determine if one cached response can be used to serve another application's request. As used herein, the term “heterogenous” means “out of its origin,” and in context of distributed caching system, it means that the cached response being served to “an application that is out of its original application” in addition to or in lieu of its own, original application; for example, a cached response for Application-A being served to Application B (e.g., providing that they are determined to be in the same category and thus suitable for this technique).

More specifically, according to some embodiments, heterogenous cache service module **490** can identify content that could be served from cache during the aggressive caching, even if it is not being cached at the moment. In some embodiments, heterogenous cache service module **490** can store a previously cached content, known now to be stale, to be served at a later date. Some embodiments of heterogenous cache service module **490** can identify requests that are substantially the same, whose responses could be used to serve another request. Further, heterogenous cache service module **490** can identify request types for which we can safely serve the previous response before aggressive caching started. In one embodiment, heterogenous cache service module **490** may include submodules, such as a cache categorization module **490**A for categorizing cached data and an application categorization module **490**B for categorizing applications.

Delaying Long Poll Responses

The present embodiments recognize that it can be beneficial to delay delivery of long poll responses from the network to slow down interaction between the application and the content server. The present embodiments further recognize that, even without caching, it is possible to slow down the interaction between application and server by holding on to a response from the server—we can know how long the application waits either by observing past pattern, or by reading the information from the request that the application makes—as it does convey this information to the server, so that the server knows when it must respond to not time out the application. This technique can serve, for example, as an addition or an alternative to aggressive caching.

Accordingly, in some embodiments, a long poll response delay module **495** can identify the longest possible delay by using a protocol parsing module **495**A to parse the protocol (the request typically has the long poll delay the application is willing to wait). In some embodiments, long poll response delay module **495** can identify the longest possible delay by using a previous transactions observing module **495**B to observe the previous transactions to find the longest successful long poll and using its length to imply the longest possible delay.

Then, long poll response delay module **495** can tune shorter this delay based on the situation to provide different levels of aggressiveness. Similarly, the aforementioned aggressive caching techniques can be tuned less aggressive by defining maximum period of postponing processing of an invalidation, or observing activity in some of the output interfaces (such as a notification LED or sound system) when new content is delivered.

It is worth noting that the present embodiments acknowledge that there may be little value in delaying long poll responses when aggressive caching is in use—as aggressive caching does capture almost all of the benefits already. Thus, when the aggressive caching is operational, long poll response delay module **495** can function as a "friendlier" alternative option for aggressive caching of long polls—there delaying the responses would provide a solution where new data from long polls gets delayed less than with aggressive caching. Similar effect could be achieved by introducing some maximum delay to processing the invalidation.

Nonetheless, when the screen is ON and aggressive caching is not operational, delaying the long poll responses (and serving cached responses) would provide significant optimization improvements. However, this would be directly visible to the user as well, as the user is actively using the device, but new data is delayed. In some embodiments, this technique can be implemented as a part of congestion management (as compared to being used as a global optimization policy).

As such, this technique can be seen as extending aggressive caching to when the screen is on. By serving user stale data from cache, carriers can benefit from this technique as it provides a way to alleviate congestion and keep as many users as satisfactory as possible.

As mentioned, long poll response delay module **495** can go hand in hand with congestion management offering and it could be offered as one option for managing congestion with other options being aggressive caching and maybe going as far as blocking apps.

Note that, according to the embodiments disclosed herein, this technique applies delaying on client only; it does not consider delaying at the server (e.g., proxy-server) which although valid and probably beneficial would need to have additional considerations, as the server doesn't know whether the user is active or not.

Server-Side Proxy

FIG. **5**A depicts a block diagram illustrating an example of server-side components in a distributed proxy and cache system, further including an extended caching optimization manager.

FIG. **5**B depicts a block diagram illustrating additional components in the extended caching optimization manager shown in the example of FIG. **5**A.

Some embodiments of the ECO manager, such as optimization manager **501** as illustrated in FIG. **5**B, can function as a complementary part to the ECO engine **401** of FIG. **4**A. For example, the optimization manager **501** can include a device specific information module **530** to receive information (e.g., as gathered by modules **402**-**406** of FIG. **4**B) that are specific to the device, and the cache freshness determination module **520** can decide whether to refresh cache (e.g., from a third-party server) and/or whether to feed a certain data to the client-proxy based on device specific information, the actual freshness of the data in the cache **335**, and the ECO setting level **510**, which can be a uniform setting across all clients served by the server-proxy, or can be device specific, application specific, user specific, group specific, or any suitable combination of above.

Alternatively, the optimization manager **501** can have similar modules as the optimization engine **401** and functions in a similar way as described above with respect to FIGS. **4**A-**4**B. In some embodiments, ECO manager **501** may include a safeguard module **540** which includes safeguard functionalities and a DNS CE expiration module **550** that caches DNS requests more aggressively to prevent unnecessary radio power-ups and activations triggered by those DNS requests.

FIG. **6** shows a diagrammatic representation of a machine in the example form of a computer system within which a set of instructions, for causing the machine to perform any one or more of the methodologies discussed herein, may be executed.

In the example of FIG. **6**, the computer system **600** includes a processor **602**, memory **604**, non-volatile memory **606**, and a network interface device **608** for communicating with a network **610**. Various common components (e.g., cache memory) are omitted for illustrative simplicity. The computer system **600** is intended to illustrate a hardware device on which any of the components depicted and/or described in this specification can be implemented. The computer system **600** can be of any applicable known or convenient type. The components of the computer system **600** can be coupled together via a bus or through some other known or convenient device.

Processor **602** may be, for example, a conventional microprocessor such as an Intel Pentium microprocessor or Motorola power PC microprocessor. One of skill in the relevant art will recognize that the terms "machine-readable (storage) medium" or "computer-readable (storage) medium" include any type of device that is accessible by the processor.

Memory **604** may be coupled to processor **602** by, for example, a bus **612**. Memory **604** can include, by way of example but not limitation, random access memory (RAM), such as dynamic RAM (DRAM) and static RAM (SRAM). The memory can be local, remote, or distributed.

Bus **612** may also couple processor **602** to non-volatile memory **606** and to a drive unit **614**. Non-volatile memory **606** may be, for example, a magnetic floppy or hard disk, a magnetic-optical disk, an optical disk, a read-only memory (ROM), such as a CD-ROM, EPROM, or EEPROM, a magnetic or optical card, or another form of storage for large amounts of data. Some of this data is often written, by a direct memory access process, into memory during execution of software in the computer **600**. The non-volatile storage can be local, remote, or distributed. Non-volatile memory **606** is optional because systems can be created with all applicable data available in memory. A typical computer system may include at least a processor, memory, and a device (e.g., a bus) coupling the memory to the processor.

Software is typically stored in the non-volatile memory and/or the drive unit, such as in a machine-readable (storage) medium **616**. Software usually includes a set of instructions **618** that cause processor **602** to perform specific tasks. Indeed, for large programs, it may not even be possible to store the entire program in the memory. Nevertheless, it should be understood that for software to run, if necessary, it is moved to a computer readable location appropriate for processing, and for illustrative purposes, that location is referred to as "memory". Thus, instructions **618** may be moved into instruction memory **620** within processor **602**, instruction memory **622** within main memory **604**, or both. Even when software is moved to the memory for execution, the processor typically make use of hardware registers to store values associated with the software, and local cache that, ideally, serves to speed up execution. As used herein, a software program is assumed to be stored at any known or convenient location (from non-volatile storage to hardware registers) when the software program is referred to as "implemented in a computer-readable medium." A processor is considered to be "configured to execute a program" when at least one value associated with the program is stored in a register readable by the processor.

The bus also couples processor **602** to network interface device **608**. The interface can include one or more of a modem or network interface. It will be appreciated that a modem or network interface can be considered to be part of the computer system. The interface can include an analog modem, isdn modem, cable modem, token ring interface, satellite transmission interface (e.g. "direct PC"), or other interfaces for coupling a computer system to other computer systems. The interface can include one or more input and/or output devices. The I/O devices can include, by way of example but not limitation, a keyboard, a mouse or other pointing device, disk drives, printers, a scanner, and other input and/or output devices, including a display device. The display device can include, by way of example but not limitation, a cathode ray tube (CRT), liquid crystal display (LCD), or some other applicable known or convenient display device. For simplicity, it is assumed that controllers of any devices not depicted in the example of FIG. **8** reside in the interface.

In one embodiment, computer system **600** may include one or more of the following: a video display **624** (e.g., a screen or monitor), an alphanumeric input device **626** (e.g., a keyboard), a cursor control device **628** (e.g., a mouse or touch screen), and a signal generator **630** (e.g., a speaker or audio output).

In operation, the computer system **600** can be controlled by operating system software that includes a file management system, such as a disk operating system. One example of operating system software with associated file management system software is the family of operating systems known as Windows® from Microsoft Corporation of Redmond, Wash., and their associated file management systems. Another example of operating system software with its associated file management system software is the Linux operating system and its associated file management system. The file management system is typically stored in the non-volatile memory and/or drive unit and causes the processor to execute the various acts required by the operating system to input and output data and to store data in the memory, including storing files on the non-volatile memory and/or drive unit.

Some portions of the detailed description may be presented in terms of algorithms and symbolic representations of operations on data bits within a computer memory. These algorithmic descriptions and representations are the means used by those skilled in the data processing arts to most effectively convey the substance of their work to others skilled in the art. An algorithm is here, and generally, conceived to be a self-consistent sequence of operations leading to a desired result. The operations are those requiring physical manipulations of physical quantities. Usually, though not necessarily, these quantities take the form of electrical or magnetic signals capable of being stored, transferred, combined, compared, and otherwise manipulated. It has proven convenient at times, principally for reasons of common usage, to refer to these signals as bits, values, elements, symbols, characters, terms, numbers, or the like.

It should be borne in mind, however, that all of these and similar terms are to be associated with the appropriate physical quantities and are merely convenient labels applied to these quantities. Unless specifically stated otherwise as apparent from the following discussion, it is appreciated that throughout the description, discussions utilizing terms such as "processing" or "computing" or "calculating" or "determining" or "displaying" or the like, refer to the action and processes of a computer system, or similar electronic computing device, that manipulates and transforms data represented as physical (electronic) quantities within the computer system's registers and memories into other data similarly represented as physical quantities within the computer system memories or registers or other such information storage, transmission or display devices.

The algorithms and displays presented herein are not inherently related to any particular computer or other apparatus. Various general purpose systems may be used with programs in accordance with the teachings herein, or it may prove convenient to construct more specialized apparatus to perform the methods of some embodiments. The required structure for a variety of these systems will appear from the description below. In addition, the techniques are not described with reference to any particular programming language, and various embodiments may thus be implemented using a variety of programming languages.

In alternative embodiments, the machine operates as a standalone device or may be connected (e.g., networked) to other machines. In a networked deployment, the machine may operate in the capacity of a server or a client machine in a client-server network environment, or as a peer machine in a peer-to-peer (or distributed) network environment.

The machine may be a server computer, a client computer, a personal computer (PC), a tablet PC, a laptop computer, a set-top box (STB), a personal digital assistant (PDA), a cellular telephone, an iPhone, a Blackberry, a processor, a telephone, a web appliance, a network router, switch or bridge, or any machine capable of executing a set of instructions (sequential or otherwise) that specify actions to be taken by that machine.

While the machine-readable medium or machine-readable storage medium is shown in an exemplary embodiment to be a single medium, the term "machine-readable medium" and "machine-readable storage medium" should be taken to include a single medium or multiple media (e.g., a centralized or distributed database, and/or associated caches and servers) that store the one or more sets of instructions. The term "machine-readable medium" and "machine-readable storage medium" shall also be taken to include any medium that is capable of storing, encoding or carrying a set of instructions for execution by the machine and that cause the machine to perform any one or more of the methodologies of the presently disclosed technique and innovation.

In general, the routines executed to implement the embodiments of the disclosure, may be implemented as part of an operating system or a specific application, component, program, object, module or sequence of instructions referred to as "computer programs." The computer programs typically comprise one or more instructions set at various times in various memory and storage devices in a computer, and that, when read and executed by one or more processing units or processors in a computer, cause the computer to perform operations to execute elements involving the various aspects of the disclosure.

Moreover, while embodiments have been described in the context of fully functioning computers and computer systems, those skilled in the art will appreciate that the various embodiments are capable of being distributed as a program product in a variety of forms, and that the disclosure applies equally regardless of the particular type of machine or computer-readable media used to actually effect the distribution.

Further examples of machine-readable storage media, machine-readable media, or computer-readable (storage) media include but are not limited to recordable type media such as volatile and non-volatile memory devices, floppy and other removable disks, hard disk drives, optical disks (e.g., Compact Disk Read-Only Memory (CD ROMS), Digital Versatile Disks, (DVDs), etc.), among others, and transmission type media such as digital and analog communication links.

FIG. **7** depicts a flowchart illustrating an exemplary process for distributed caching of information using extended caching optimization according to an embodiment of the subject matter described herein. In the embodiment illustrated in FIG. **7**, the method includes, at a mobile device for operating in a wireless network, the device including a local proxy for monitoring requests issued from an application located within the device to an external entity not located within the device and for storing, in a local cache, responses to the monitored requests received from the external entity, identifying a request as one that meets a first criterion for optimization (step **700**).

In response to identifying the request as one that meets a first criterion for optimization, an extended caching optimization is applied (step **702**). Applying an extended caching optimization includes preventing the identified request from being transmitted to the external entity and providing a response to the identified request from the local cache.

Unless the context clearly requires otherwise, throughout the description and the claims, the words "comprise," "comprising," and the like are to be construed in an inclusive sense, as opposed to an exclusive or exhaustive sense; that is to say, in the sense of "including, but not limited to." As used herein, the terms "connected," "coupled," or any variant thereof, means any connection or coupling, either direct or indirect, between two or more elements; the coupling of connection between the elements can be physical, logical, or a combination thereof. Additionally, the words "herein," "above," "below," and words of similar import, when used in this application, shall refer to this application as a whole and not to any particular portions of this application. Where the context permits, words in the above Detailed Description using the singular or plural number may also include the plural or singular number respectively. The word "or," in reference to a list of two or more items, covers all of the following interpretations of the word: any of the items in the list, all of the items in the list, and any combination of the items in the list.

The above detailed description of embodiments of the disclosure is not intended to be exhaustive or to limit the teachings to the precise form disclosed above. While specific embodiments of, and examples for, the disclosure are described above for illustrative purposes, various equivalent modifications are possible within the scope of the disclosure, as those skilled in the relevant art will recognize. For example, while processes or blocks are presented in a given order, alternative embodiments may perform routines having steps, or employ systems having blocks, in a different order, and some processes or blocks may be deleted, moved, added, subdivided, combined, and/or modified to provide alternative or subcombinations. Each of these processes or blocks may be implemented in a variety of different ways. Also, while processes or blocks are at times shown as being performed in series, these processes or blocks may instead be performed in parallel, or may be performed at different times. Further any specific numbers noted herein are only examples: alternative implementations may employ differing values or ranges.

The teachings of the disclosure provided herein can be applied to other systems, not necessarily the system described above. The elements and acts of the various embodiments described above can be combined to provide further embodiments.

Any patents and applications and other references noted above, including any that may be listed in accompanying filing papers, are incorporated herein by reference. Aspects of the disclosure can be modified, if necessary, to employ the systems, functions, and concepts of the various references described above to provide yet further embodiments of the disclosure.

These and other changes can be made to the disclosure in light of the above Detailed Description. While the above description describes certain embodiments of the disclosure, and describes the best mode contemplated, no matter how detailed the above appears in text, the teachings can be practiced in many ways. Details of the system may vary considerably in its implementation details, while still being encompassed by the subject matter disclosed herein. As noted above, particular terminology used when describing certain features or aspects of the disclosure should not be taken to imply that the terminology is being redefined herein to be restricted to any specific characteristics, features, or aspects of the disclosure with which that terminology is associated. In general, the terms used in the following claims should not be construed to limit the disclosure to the specific embodiments disclosed in the specification, unless the above Detailed Description section explicitly defines such terms. Accordingly, the actual scope of the disclosure encompasses not only the disclosed embodiments, but also all equivalent ways of practicing or implementing the disclosure under the claims.

While certain aspects of the disclosure are presented below in certain claim forms, the inventors contemplate the various aspects of the disclosure in any number of claim forms. For example, while only one aspect of the disclosure is recited as a means-plus-function claim under 35 U.S.C. §112, ¶ 6, other aspects may likewise be embodied as a means-plus-function claim, or in other forms, such as being embodied in a computer-readable medium. (Any claim intended to be treated under 35 U.S.C. §112, ¶ 6 begins with the words "means for".) Accordingly, the applicant reserves the right to add additional claims after filing the application to pursue such additional claim forms for other aspects of the disclosure.

## Embodiments

The subject matter disclosed herein includes, but is not limited to, the following embodiments:

1. A method for distributed caching of information using extended caching optimization, the method including, at a mobile device for operating in a wireless network, the device including a local proxy for monitoring requests issued from an application located within the device to an external entity not located within the device and for storing, in a local cache, responses to the monitored requests received from the external entity, identifying a request as one that meets a first criterion for optimization, and, in response to such identification, applying an extended caching optimization. Applying an extended caching optimization includes preventing the identified request from being transmitted to the external entity, and providing a response to the identified request from the local cache.

2. The method of embodiment 1 where the first criterion for optimization includes a determination that the request matches an observed periodicity.

3. The method of embodiment 2 where the observed periodicity comprises a simple periodicity or a complex periodicity.

4. The method of embodiment 1 where, responsive to a detection by the local proxy of a repetition of a request-response interaction between the application and the local proxy that exceeds a threshold rate, applying a traffic calming technique to signaling messages related to the application.

5. The method of embodiment 4 where applying the traffic calming technique includes increasing, by the local proxy, a delay between receiving the request from the application and providing the cached response to the application.

6. The method of embodiment 4 including, responsive to a detection that the application terminates a connection to the external entity as a result of an increased delay between sending the request and receiving the cached response, decreasing, by the local proxy, the delay between receiving a request from the application and providing a cached response to the application.

7. The method of embodiment 6 including, responsive to a determination that the application has terminated the connection to the external entity more than a threshold number of times, terminating the traffic calming technique being applied to signaling messages related to the application.

8. The method of embodiment 1 where, responsive to a detection by the local proxy that the application is requesting content that has been invalidated, terminating the extended caching optimization for that application.

9. The method of embodiment 4 including maintaining, by the local proxy, a value of the shortest request interval (RI) between successive requests from the application.

10. The method of embodiment 9 including using the maintained value of the shortest RI to identify a non-periodic request as being associated with the application.

11. The method of embodiment 4 including maintaining, by the local proxy, a value of the longest response delay (RD) observed between a request from the application and a response to the request from the external entity.

12. The method of embodiment 11 including using the maintained value of the longest RD to identify a request/response interaction as a long-poll interaction for the application.

13. The method of embodiment 12 including using the maintained value of the longest RD as the polling frequency during extended caching optimization for the application.

14. The method of embodiment 1 where the first criterion for optimization includes a determination that the request occurred while the mobile device is in a target state.

15. The method of embodiment 14 where the target state includes a display of the mobile device is not lit, a radio circuit of the mobile device is not active, a power consumption of the mobile device is below a threshold value, and/or an absence of user activity for longer than a threshold period.

16. The method of embodiment 14 where the first criterion for optimization is selectable by a user of the mobile device.

17. The method of embodiment 1 where the identified request comprises a domain name service (DNS) request.

18. The method of embodiment 17 including not terminating the extended caching optimization of the DNS request until after a threshold time to live (TTL).

19. The method of embodiment 17 including not terminating the extended caching optimization of the DNS request until after a threshold time to live (TTL) and a determination that the screen is lit.

20. The method of embodiment 17 including not terminating the extended caching optimization of the DNS request until after a threshold time to live (TTL), a determination that the screen is lit, and a determination that the radio transceiver is powered on or activated.

21. The method of embodiment 1 where, responsive to a detection by the local proxy of a repetition of a request-response interaction between the application and the local proxy that exceeds a threshold rate, applying a traffic calming technique to signaling messages related to the application.

22. The method of embodiment 1 where identification of a request as one that meets a first criterion for optimization includes identification based on a characteristic of the request, a characteristic of the requested data, and/or a characteristic of the application making the request.

23. The method of embodiment 1 including, at a proxy server located external to the mobile device and for receiving requests from the mobile device, forwarding the requests to content or service providers, receiving from the content or service providers responses to the forwarded requests, and providing the responses back to the mobile device: storing, in a server cache, copies of the forwarded requests, storing, in the server cache, copies of the received responses to the forwarded requests, and responsive to identifying a request/response pair as one that meets a second criterion for optimization, preventing the response from being provided back to the mobile device.

24. The method of embodiment 23 where, for each request/response pair identified as meeting a criterion for optimization, the method includes issuing, by the proxy server, a subsequent request to the content or service provider on behalf of the mobile device, receiving a reply to the subsequent request, and determining whether to provide or not provide the reply back to the mobile device and providing or not providing the reply according to the determination.

25. The method of embodiment 24 where providing or not providing the reply according to the determination includes

providing the reply to mobile device responsive to a determination that the reply differs from a received response stored in the server cache, a determination that the reply has a high priority, and/or a determination that the reply includes time critical data.

26. The method of embodiment 23 including, at the proxy server, responsive to a determination that the reply differs from a received response stored in the server cache, indicating to the local proxy that the cached resource has changed.

27. The method of embodiment 26 where the local cache is refreshed from the external entity immediately.

28. The method of embodiment 26 where the local cache is not refreshed from the external entity until the screen is lit.

29. The method of embodiment 26 where the local cache is not refreshed from the external entity until the screen is lit and the radio transceiver is powered on or activated.

30. The method of embodiment 26 including, at the local proxy, responsive to receipt of the indication that cached resource has changed, invalidating the cached resource immediately, invalidating the cached resource when the screen is lit, invalidating the cached resource when the screen is lit and the radio transceiver is powered on or activated, and/or ignoring the indication.

31. The method of embodiment 30 further including instructing the application to request and receive data from the external entity.

32. The method of embodiment 31 including providing a response to the request from the local cache instead of from the external entity.

33. A system for distributed caching of information using extended caching optimization, the system including a mobile device for operating in a wireless network, where the device includes a local cache and a local proxy for monitoring requests issued from an application located within the device to an external entity not located within the device and for storing, in the local cache, responses to the monitored requests received from the external entity, where the local proxy is configured to identify a request as one that meets a first criterion for optimization and apply an extended caching optimization, and where applying an extended caching optimization includes preventing the identified request from being transmitted to the external entity, and providing a response to the identified request from the local cache.

34. The system of embodiment 33 where the first criterion for optimization includes a determination that the request matches an observed periodicity.

35. The system of embodiment 33 where the observed periodicity comprises a simple periodicity or a complex periodicity.

36. The system of embodiment 33 where the local proxy is configured to detect a repetition of a request-response interaction between the application and the local proxy that exceeds a threshold rate and apply a traffic calming technique to signaling messages related to the application.

37. The system of embodiment 36 where applying the traffic calming technique includes increasing, by the local proxy, a delay between receiving the request from the application and providing the cached response to the application.

38. The system of embodiment 37 where, responsive to a detection that the application terminates a connection to the external entity as a result of an increased delay between sending the request and receiving the cached response, the local proxy decreases the delay between receiving a request from the application and providing a cached response to the application.

39. The system of embodiment 38 where, responsive to a determination that the application has terminated the connection to the external entity more than a threshold number of times, the local proxy terminates the traffic calming technique being applied to signaling messages related to the application.

40. The system of embodiment 33 where, responsive to a detection that the application is requesting content that has been invalidated, the local proxy terminates the extended caching optimization for that application.

41. The system of embodiment 36 where the local proxy maintains a value of the shortest request interval (RI) between successive requests from the application.

42. The system of embodiment 41 where the local proxy uses the maintained value of the shortest RI to identify a non-periodic request as being associated with the application.

43. The system of embodiment 36 where the local proxy maintains a value of the longest response delay (RD) observed between a request from the application and a response to the request from the external entity.

44. The system of embodiment 43 where the local proxy uses the maintained value of the longest RD to identify a request/response interaction as a long-poll interaction for the application.

45. The system of embodiment 44 where the local proxy uses the maintained value of the longest RD as the polling frequency during extended caching optimization for the application.

46. The system of embodiment 33 where the first criterion for optimization includes a determination that the request occurred while the mobile device is in a target state.

47. The system of embodiment 46 where the target state includes a display of the mobile device is not lit, a radio circuit of the mobile device is not active, a power consumption of the mobile device is below a threshold value, and/or an absence of user activity for longer than a threshold period.

48. The system of embodiment 46 where the first criterion for optimization is selectable by a user of the mobile device.

49. The system of embodiment 33 where the identified request comprises a domain name service (DNS) request.

50. The system of embodiment 49 where the local proxy does not terminate the extended caching optimization of the DNS request until after a threshold time to live (TTL).

51. The system of embodiment 49 where the local proxy does not terminate the extended caching optimization of the DNS request until after a threshold time to live (TTL) and a determination that the screen is lit.

52. The system of embodiment 49 where the local proxy does not terminate the extended caching optimization of the DNS request until after a threshold time to live (TTL), a determination that the screen is lit, and a determination that the radio transceiver is powered on or activated.

53. The system of embodiment 33 where, responsive to a detection of a repetition of a request-response interaction between the application and the local proxy that exceeds a threshold rate, the local proxy applies a traffic calming technique to signaling messages related to the application.

54. The system of embodiment 33 where identification of a request as one that meets a first criterion for optimization includes identification based on a characteristic of the request, a characteristic of the requested data, and/or a characteristic of the application making the request.

55. The system of embodiment 33 including a proxy server, located external to the mobile device, that receives requests from the mobile device, forwards the requests to

content or service providers, receives from the content or service providers responses to the forwarded requests, provides the responses back to the mobile device, stores, in a server cache, copies of the forwarded requests, stores, in the server cache, copies of the received responses to the forwarded requests, identifies a request/response pair as one that meets a second criterion for optimization, and, responsive to identifying a request/response pair as one that meets a second criterion for optimization, prevents the response from being provided back to the mobile device.

56. The system of embodiment 55 where, for each request/response pair identified as meeting a criterion for optimization, the proxy server issues a subsequent request to the content or service provider on behalf of the mobile device, receives a reply to the subsequent request, determines whether to provide or not provide the reply back to the mobile device, and provides or not provides the reply according to the determination.

57. The system of embodiment 56 where the proxy server provides or not provides the reply based on a determination that the reply differs from a received response stored in the server cache, a determination that the reply has a high priority, and/or a determination that the reply includes time critical data.

58. The system of embodiment 55 where, responsive to a determination that the reply differs from a received response stored in the server cache, the proxy server indicates to the local proxy that the cached resource has changed.

59. The system of embodiment 58 where, in response to receiving from the proxy server the indication that the cached resource has changed, the local cache is refreshed from the external entity immediately.

60. The system of embodiment 58 where, in response to receiving from the proxy server the indication that the cached resource has changed, the local cache is not refreshed from the external entity until the screen is lit.

61. The system of embodiment 58 where, in response to receiving from the proxy server the indication that the cached resource has changed, the local cache is not refreshed from the external entity until the screen is lit and the radio transceiver is powered on or activated.

62. The system of embodiment 58 where the local proxy, responds to receipt of the indication that cached resource has changed by invalidating the cached resource immediately, invalidating the cached resource when the screen is lit, invalidating the cached resource when the screen is lit and the radio transceiver is powered on or activated, or ignoring the indication.

63. The system of embodiment 62 where the local proxy instructs the application to request and receive data from the external entity.

64. The system of embodiment 62 where the local proxy provides a response to the request from the local cache instead of from the external entity.

What is claimed is:

**1**. A method for distributed caching of information using extended caching optimization, the method comprising:
- at a mobile device for operating in a wireless network, the device including a local proxy for monitoring requests issued from an application located within the device to an external entity not located within the device and for storing, in a local cache, responses to the monitored requests received from the external entity:
  - identifying a request as one that meets a first criterion for optimization; and
  - in response to identifying the request as one that meets a first criterion for optimization, applying an extended caching optimization, wherein applying an extended caching optimization includes:
    - preventing the identified request from being transmitted to the external entity; and
    - providing a response to the identified request from the local cache.

**2**. The method of claim **1** wherein the first criterion for optimization includes a determination that the request matches an observed periodicity.

**3**. The method of claim **2** wherein the observed periodicity comprises a simple periodicity or a complex periodicity.

**4**. The method of claim **1** wherein, responsive to a detection by the local proxy of a repetition of a request-response interaction between the application and the local proxy that exceeds a threshold rate, applying a traffic calming technique to signaling messages related to the application.

**5**. The method of claim **4** wherein applying the traffic calming technique includes increasing, by the local proxy, a delay between receiving the request from the application and providing the cached response to the application.

**6**. The method of claim **4** including, responsive to a detection that the application terminates a connection to the external entity as a result of an increased delay between sending the request and receiving the cached response, decreasing, by the local proxy, the delay between receiving a request from the application and providing a cached response to the application.

**7**. The method of claim **6** including, responsive to a determination that the application has terminated the connection to the external entity more than a threshold number of times, terminating the traffic calming technique being applied to signaling messages related to the application.

**8**. The method of claim **1** wherein, responsive to a detection by the local proxy that the application is requesting content that has been invalidated, terminating the extended caching optimization for that application.

**9**. The method of claim **4** including maintaining, by the local proxy, a value of the shortest request interval (RI) between successive requests from the application.

**10**. The method of claim **9** including using the maintained value of the shortest RI to identify a non-periodic request as being associated with the application.

**11**. The method of claim **4** including maintaining, by the local proxy, a value of the longest response delay (RD) observed between a request from the application and a response to the request from the external entity.

**12**. The method of claim **11** including using the maintained value of the longest RD to identify a request/response interaction as a long-poll interaction for the application.

**13**. The method of claim **12** including using the maintained value of the longest RD as the polling frequency during extended caching optimization for the application.

**14**. The method of claim **1** wherein the first criterion for optimization is selectable by a user of the mobile device.

**15**. A system for distributed caching of information using extended caching optimization, the system including:
- a mobile device for operating in a wireless network, the device including:
  - a local cache; and
  - a local proxy for monitoring requests issued from an application located within the device to an external entity not located within the device and for storing, in the local cache, responses to the monitored requests received from the external entity,

wherein the local proxy is configured to identify a request as one that meets a first criterion for optimization and apply an extended caching optimization, and

wherein applying an extended caching optimization includes preventing the identified request from being transmitted to the external entity, and providing a response to the identified request from the local cache.

**16**. The system of claim **15** wherein the first criterion for optimization includes a determination that the request matches an observed periodicity.

**17**. The system of claim **15** wherein the observed periodicity comprises a simple periodicity or a complex periodicity.

**18**. The system of claim **15** wherein the local proxy is configured to detect a repetition of a request-response interaction between the application and the local proxy that exceeds a threshold rate and apply a traffic calming technique to signaling messages related to the application.

**19**. The system of claim **18** wherein applying the traffic calming technique includes increasing, by the local proxy, a delay between receiving the request from the application and providing the cached response to the application.

**20**. The system of claim **19** wherein, responsive to a detection that the application terminates a connection to the external entity as a result of an increased delay between sending the request and receiving the cached response, the local proxy decreases the delay between receiving a request from the application and providing a cached response to the application.

**21**. The system of claim **20** wherein, responsive to a determination that the application has terminated the connection to the external entity more than a threshold number of times, the local proxy terminates the traffic calming technique being applied to signaling messages related to the application.

**22**. The system of claim **15** wherein, responsive to a detection that the application is requesting content that has been invalidated, the local proxy terminates the extended caching optimization for that application.

**23**. The system of claim **18** wherein the local proxy maintains a value of the shortest request interval (RI) between successive requests from the application.

**24**. The system of claim **23** wherein the local proxy uses the maintained value of the shortest RI to identify a non-periodic request as being associated with the application.

**25**. The system of claim **18** wherein the local proxy maintains a value of the longest response delay (RD) observed between a request from the application and a response to the request from the external entity.

**26**. The system of claim **25** wherein the local proxy uses the maintained value of the longest RD to identify a request/response interaction as a long-poll interaction for the application.

**27**. The system of claim **26** wherein the local proxy uses the maintained value of the longest RD as the polling frequency during extended caching optimization for the application.

**28**. The system of claim **15** wherein the first criterion for optimization is selectable by a user of the mobile device.

**29**. A non-transitory computer readable medium having stored thereon executable instructions that when executed by the processor of a computer control the computer to perform steps comprising:

identifying, at a mobile device for operating in a wireless network, the device including a local proxy for monitoring requests issued from an application located within the device to an external entity not located within the device and for storing, in a local cache, responses to the monitored requests received from the external entity, a request as one that meets a first criterion for optimization; and

in response to identifying the request as one that meets a first criterion for optimization, applying an extended caching optimization, wherein applying an extended caching optimization includes:

preventing the identified request from being transmitted to the external entity; and

providing a response to the identified request from the local cache.

* * * * *